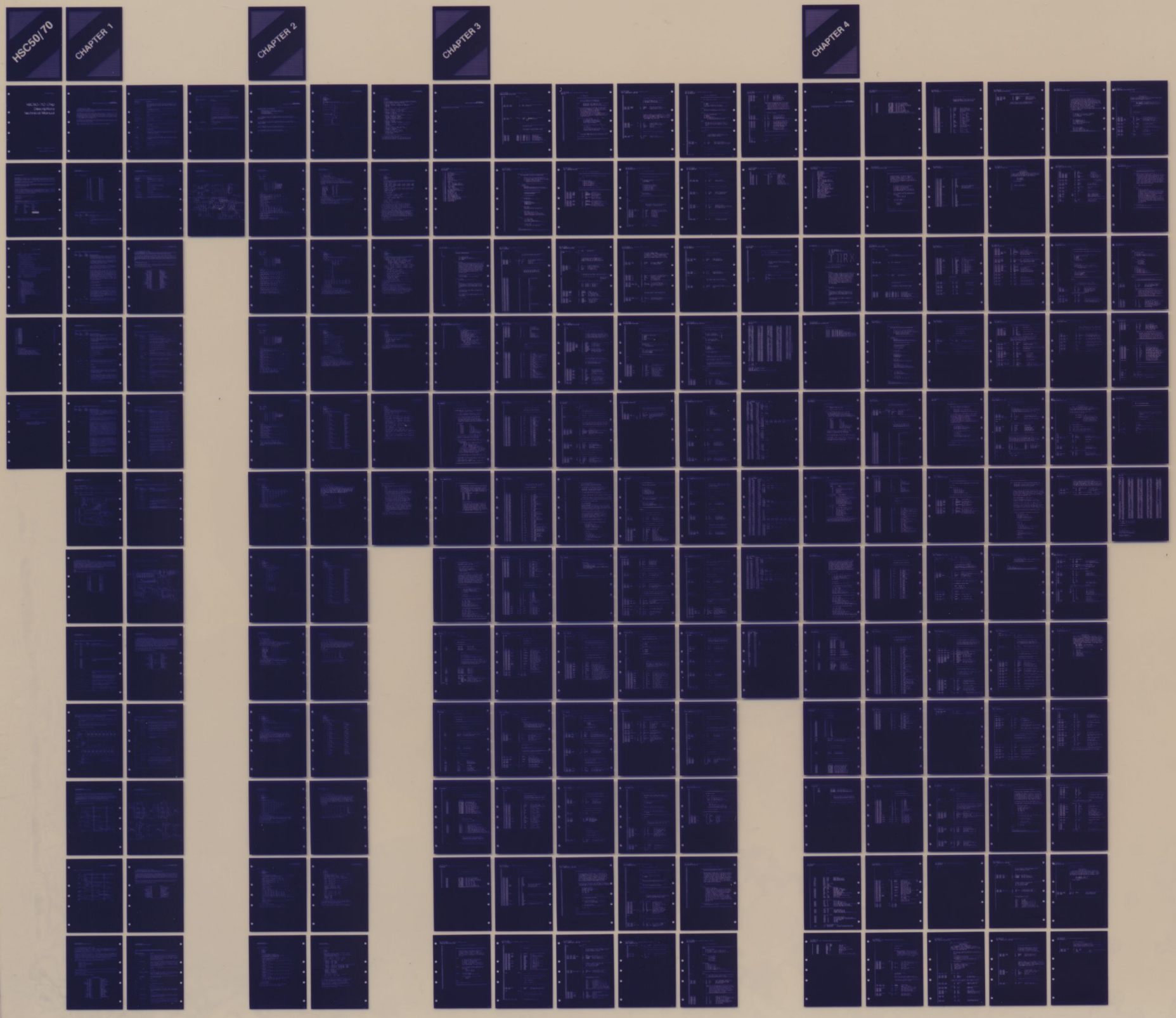
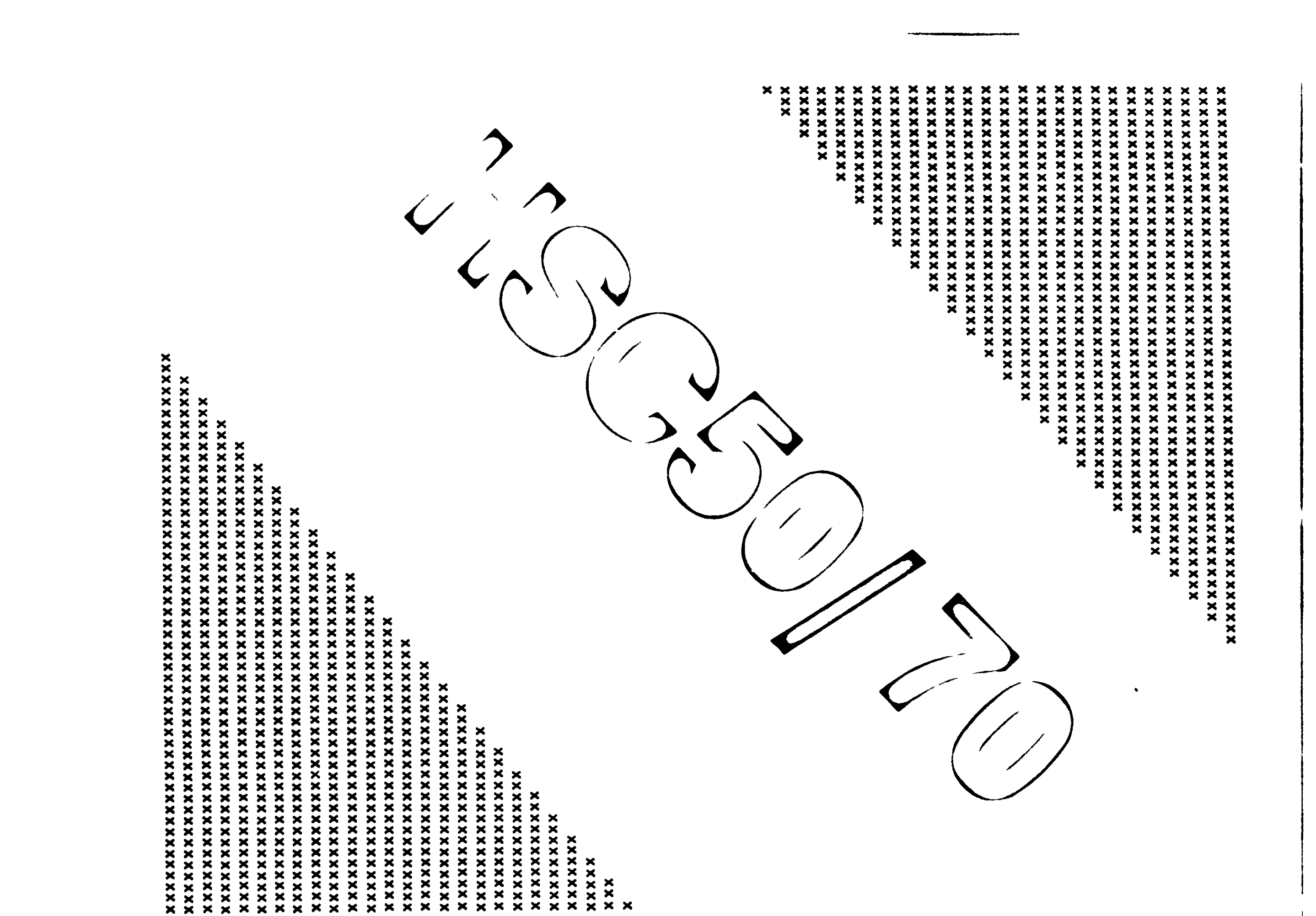
HSC50/70

CHIP DESCRIPTIONS
TECHNICAL MANUAL
FOR INTERNAL USE ONLY

EP-HS574-TM-001
1 OF 1 NOV 1987
COPYRIGHT© 1986

digital
MADE IN USA




ISSCC70

EK-HS574-TM-001

HSC50/70 Chip Descriptions Technical Manual

Prepared by Educational Services
of
Digital Equipment Corporation

First Edition, May 1986

Digital Equipment Corporation makes no representation that use of its products with those of other manufacturers will not infringe existing or future patent rights The descriptions contained herein do not imply the granting of a license to make use or sell equipment or software as described in this manual

Digital Equipment Corporation assumes no responsibility or liability for the proper performance of other manufacturers products used with its products

Digital Equipment Corporation believes that information in this publication is accurate as of its publication date Such information is subject to change without notice Digital Equipment Corporation is not responsible for any inadvertent errors

Class A Computing Devices

NOTICE This equipment generates uses, and may emit radio frequency energy It has been tested and found to comply with the limits for a Class A computing device pursuant to Subpart J of Part 15 of FCC rules for operation in a commercial environment This equipment, when operated in a residential area may cause interference to radio/TV communications In such event the user (owner) at his own expense my be required to take corrective measures

Copyright ©1986 by Digital Equipment Corporation

All Rights Reserved
Printed in U S A

The postpaid READER'S COMMENTS form on the last page of this document requests the user's critical evaluation to assist in preparing future documentation

The following are trademarks of Digital Equipment Corporation

DEC	DECnet	UDA50
DECUS	DECsystem 10	HSC
DIGITAL	DECSYSTEM 20	MASSBUS
PDP	RSX	RA80
UNIBUS	VMS	RA81
VT	RA60	KDA50-Q
RC25	TA78-81	
VAX	TU78-81	digital
RA80	KDB50	

This document was prepared using an in house documentation production system All page composition and make up was performed by TEX the typesetting system developed by Donald E Knuth at Stanford University TEX is a registered trademark of the American Mathematical Society

TABLE OF CONTENTS

Preface v

CHAPTER 1 HSC50/70 SPECIAL CHIP DESCRIPTIONS

1 1 SERDES 10 AND 16 (DC018) 1-1

1.2 Reed Solomon Generator (DC309) 1 7

1 3 REAL TIME STATE GATE ARRAY (DC703) 1-12

1 4 DUAL DESERIALIZER (DC704) 1-15

1 5 SDI/STI RECEIVER/DRIVER (DC026) 1-20

1 6 SDI/STI ENCODER/DECODER (DC027) 1-23

CHAPTER 2 HSC50/70 PAL EQUATIONS

2 1 PAL Equations for PALs on the HSC50 Input/Output Control Processor Module 2 1

2.2 PAL Equations for PALS on the HSC70 Input/Output Control Processor Module 2 1

2 3 PAL Equations for PALS on the HSC70 Memory Module 2 1

2 4 PAL Equations for PALS on the Data Channel Module 2 1

CHAPTER 3 HSC50 BOOT PROM LISTINGS

CHAPTER 4 HSC70 BOOT PROM LISTINGS

FIGURES

1-1 SERDES Pin Layout 1-2

1-2 SERDES Function Block Diagram 1 6

1-3 Reed Solomon Generator Pinout 1 7

1-4 R-S GEN Functional Block Diagram 1 9

1-5 R-S GEN Typical Write Operation 1 10

1-6 R-S GEN Typical Read Operation 1-11

1-7 Real Time State Gate Array Pinout 1 12

1-8 Dual Deserializer Pin Layout 1 15

1-9 Dual Deserializer Functional Block Diagram 1-19

1-10 SDI/STI Receiver/Driver Pin Layout 1 20

1-11 SDI/STI Receiver/Driver Functional Block Diagram 1 22

1-12 SDI/STI Encoder/Decoder Pin Layout 1-23

1-13 SDI/STI Encoder/Decoder Functional Block Diagram 1 26

2-1 23-062J5-00 2 2

2-2 23-063J5 00) 2 3

2-3 23-064J5-00) 2 4

2-4 23-065J5 00) 2 5

2-5 23-135K5-00) 2 6

2-6 23-161K3 00) 2 7

2-7 23-258J5-00) 2 9

2-8 23-259J5 00) 2 10

2-9 23-260J5 00 2-11
2-10 23-261J5-00 2-12
2-11 23-262J5 00 2-13
2-12 23-263J5-00 2-15
2-13 23-264J5 00 2-16
2-14 23-007K7 00 2-17
2-15 23-008K7-00 2-19
2-16 23-009K7-00 2-21
2-17 23-127k5-00 2-23
2-18 23-128K5 00 2 24
2-19 23-131K5-00 2 25
2-20 23-143K5-00 2 26
2-21 23-144K5-00 2 27
2-22 23-2495 00 2 28
2-23 23-069K5-00 2 29

TABLES

1-1 SERDES Signal Definitions and Functions 1 2
1-2 Word Length Select 1-6
1-3 Read-Solomon Generator Signal Definitions And Functions 1-8
1-4 RTS Gate Array Functional Signal Description 1-13
1-5 Dual Deserializer Signal Definitions and Functions 1-16
1-6 SDI/STI Receiver/Driver Signal Definitions and Functions 1 21
1-7 SDI/STI Encoder/Decoder Signal Definitions and Functions 1 24

Preface

This manual is a support document for the *HSC50 70 Hardware Technical Manual* It includes the following topics

- Descriptions of special chips used in the HSC50/70
- PAL equations for the PALs in the HSC50/70
- HSC50 Boot PROM listings
- HSC70 Boot PROM listings

All of the informations contained in this manual is intended for the support level field service representative

NOTE

Because this manual covers both the HSC50 and HSC70, the acronym HSC is used to discuss functions common to both controllers

CHAPTER 1

CHAPTER 1
HSC50/70 SPECIAL CHIP DESCRIPTIONS

1.1 SERDES 10 AND 16 (DC018)

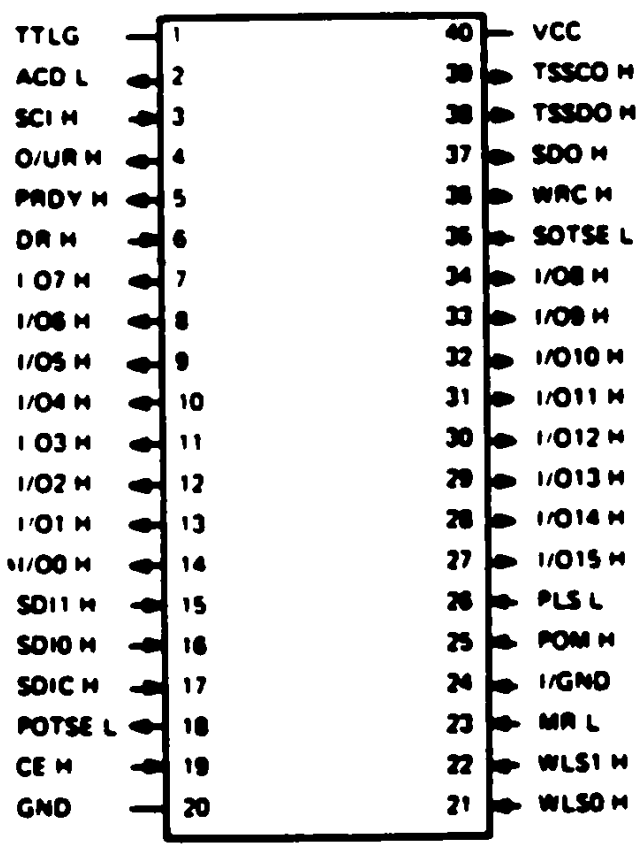
The Serializer Deserializer (SERDES) chip was specially designed by Digital Equipment Corporation to perform data conversion between serial and parallel data formats The data conversion may occur at a maximum frequency of approximately 25 MHz

The SERDES contains an automatic synchronization feature that allows it to synchronize on a serial bit stream without any external control Using an auto-correlation technique the SERDES detects a 12 bit pattern and uses this pattern as a synchronization point The auto-correlation feature permits up to 3 bits of the 12 bit pattern to be in error yet still achieve correct synchronization After this point is detected the SERDES begins the serial to parallel conversion There are three possible parallel word lengths 8-bit 10-bit (SERDES 10), and 16-bit (SERDES 16)

The SERDES has one parallel output port and two serial output ports The parallel port and one of the serial ports are tri state These tri state ports have separate external enable signals The remaining serial port is a standard TTL totem pole output

Figure 1-1 shows the pin layout of the SERDES and Table 1-1 provides a functional description of the signals on the output pins Figure 1-2 is a functional block diagram of the SERDES

Figure 1-1 SERDES Pin Layout



CX-6388

Table 1-1 SERDES Signal Definitions and Functions

Pin Number	Pin Name	Input/ Output	Signal Name/Function
1	TTL G	—	TTL GROUND
2	ACD L	Output	AUTO CORRELATOR—The SERDES asserts this signal when the sync byte is detected This signal is for test purposes only and is not used
3	SCI H	Input	SERIAL CLOCK IN—The positive edge of this signal shifts data into the shift register

Table 1-1 (Cont) SERDES Signal Definitions and Functions

Pin Number	Pin Name	Input/ Output	Signal Name/Function
4	O/UR H	Output	OVER/UNDER RUN—This signal is cleared by MASTER RESET It is asserted when a parallel transfer on the parallel port is missed The missed parallel transfer occurs in one of the following ways 1 In parallel-out mode (POM asserted) the SERDES asserts PRDY when the parallel output data is ready If a corresponding DR (data received) signal is not returned within the word boundary O/UR is asserted, indicating an overrun condition 2 In parallel in mode (POM negated), the SERDES asserts PRDY when it is ready to be parallel loaded If a corresponding PLS (parallel input strobe) is not received within the word boundary O/UR is asserted, indicating an underrun condition
5	PRDY H	Output	PARALLEL READY—NOTE The following assumes the tri-state ports are enabled (POTSE asserted) PRDY helps initiate and complete asynchronous handshaking routines between the parallel port on the SERDES and a corresponding parallel device In parallel-out mode (POM asserted), the SERDES asserts POM when the parallel output word is stable on the I/O pins However if the sync character is present on the I/O lines PRDY is not asserted In parallel-in mode (POM negated) the SERDES asserts PRDY when it is ready to receive a parallel word PRDY negates in one of the following ways 1 If POM is asserted a negative to positive transition on DR (data received) negates PRDY 2 If POM is negated, a positive-to-negative transition PLS (parallel input strobe) negates PRDY 3 If neither 1 or 2 above occurs the SERDES itself drops PRDY just before the parallel data becomes invalid and latches an O/UR
6	DR H	Input	DATA RECEIVED—In parallel out mode (POM asserted) SERDES receives this signal from the corresponding device indicating the device has received the data This is part of the asynchronous parallel handshake The handshake sequence is as follows The SERDES asserts PRDY when the parallel output data is stable After the user has taken the data it asserts DR H Upon receiving DR H the SERDES negates PRDY
7-14 and 27-34	I/O 15 00	Input/ Output	PARALLEL DATA I/O —These lines contain parallel data and their I/O function is determined by the state of POM Their output is enabled by the POTSE (parallel output tri state enable) signal
15	SDI1 H	Input	SERIAL DATA IN 1—This serial input data is clocked into the shift register if SDIC is asserted or high

Table 1-1 (Cont.) SERDES Signal Definitions and Functions

Pin Number	Pin Name	Input Output	Signal Name/Function
16	SDI0 H	Input	SERIAL DATA IN 0—This serial input data is clocked into the shift register if SDIC is negated or low
17	SDIC H	INPUT	SERIAL DATA INPUT CONTROL—This signal determines which serial input is shifted into the shift register. If it is low, SDI0 data is moved into the shift register. If it is high, SDI1 data is moved into the shift register.
18	POTSF L	Input	PARALLEL OUT TRI STATE ENABLE—If negated, this signal disables all the parallel outputs (I/O <15:00>)
19	CE H	Input	COUNTER ENABLE—With POM negated, CE enables the internal bit rate counter on the next positive edge of SCI. With POM asserted, CE permits the detection of the sync pattern in the SDI stream to start the bit rate counter. The sync pattern is as follows: Preamble 0s 0000000000 Sync Pattern 111101011001100 Data XXXXXX
20	GND	—	GROUND
21 22	WLS0 WLS1	Input Input	WORD LENGTH SELECT 0 and 1—These two select lines determine the length of the parallel word in or out of the SERDES. Table 1-2 describes the possible parallel word lengths.
23	MR L	Input	MASTER RESET—This signal resets all the internal registers and counters.
24	I/O GND	—	I/O GROUND—Isolated ground for the I/O port output buffers (I/O <15:00>)
25	POM H	Input	PARALLEL-OUT MODE—When asserted, POM causes SERDES inputs and outputs to be serial in, serial out, and parallel out. When negated, POM causes the SERDES inputs and outputs to be parallel in and serial out.
26	PLS L	Input	PARALLEL INPUT STROBE—When the SERDES is in parallel in mode (POM negated), PLS completes the parallel handshake. For example, the SERDES asserts PRDY, indicating it is ready to be parallel loaded. After the sending device has stable data on the I/O lines, it asserts PLS. This loads the SERDES registers and causes the SERDES to negate PRDY.
35	SOTSF L	Input	SERIAL OUT TRI STATE ENABLE—If negated, this signal disables the two serial tri state outputs (TSSIO and TSSO).

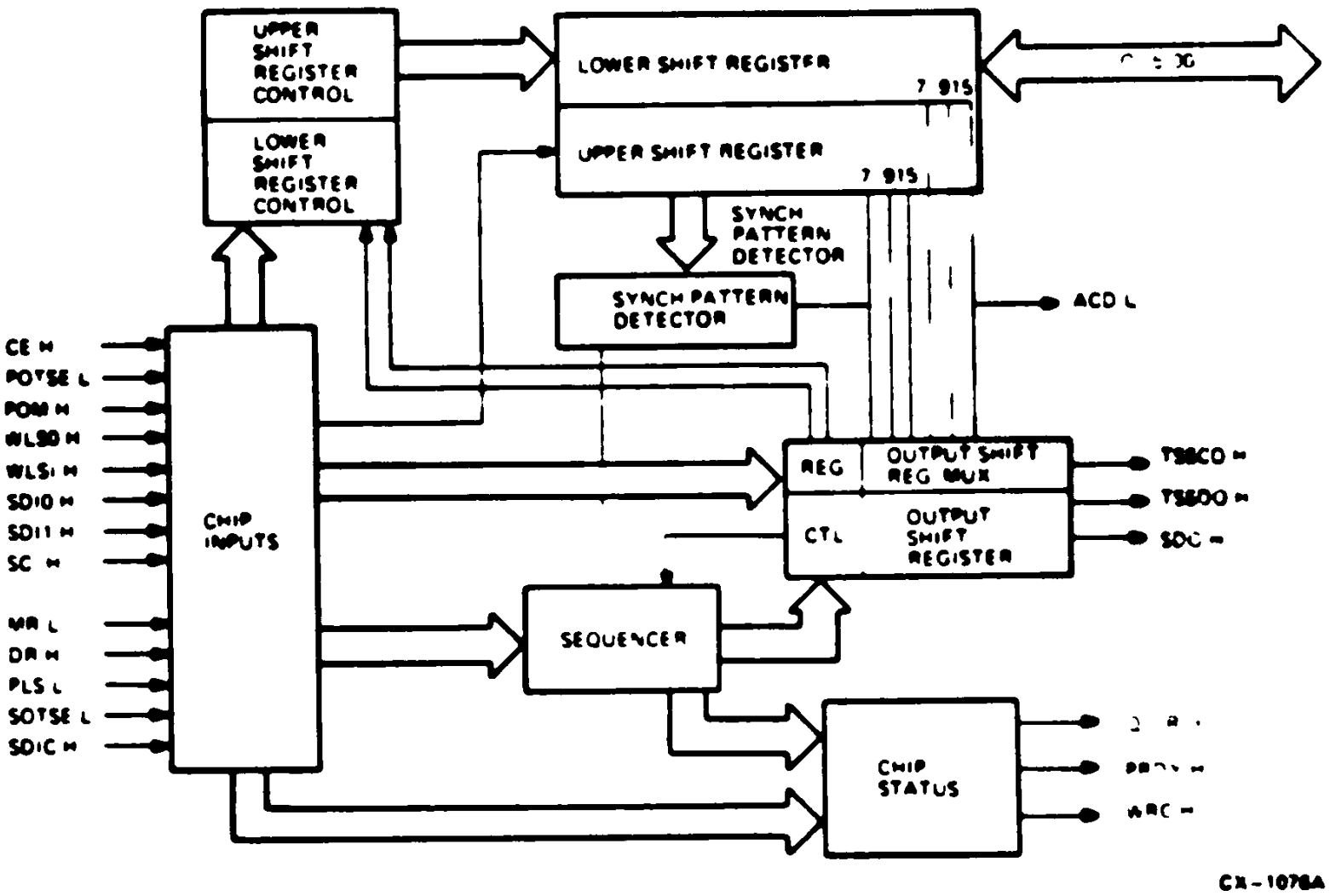
Table 1-1 (Cont.) SERDES Signal Definitions and Functions

Pin Number	Pin Name	Input Output	Signal Name/Function
36	WRC H	Output	WORD RATE CLOCK—WRC is intended to be useful in implementing the parallel port. It is decoded from the Modulo N (N = 8, N = 10, N = 16) counter. It has a 50% duty cycle and a period of N bit times. It is always asserted when the counter is equal to zero and is negated after N/2 bit times.
37	SDO H	Output	SERIAL DATA OUT—SDO has the same delay characteristics from SDI (0 or 1) and I/O <15:00> as described in TSSDO. SDO is not a tri state output and is intended for serially cascading two SERDES running on the same Serial Clock In (SCI).
38	TSSDO H	Output	TRI-STATE SERIAL DATA OUT—TSSDO is enabled by SOTSE (Serial Out Tri state Enable). Data transitions on this line are carefully matched in time to positive edges of TSSCO (Tri state Serial Clock Out). If POM is asserted, the delay from SDI0 or SDI1 to TSSDO varies with the word length selected via WLS0 and WLS1. For a word length of 16 bits, the delay is 37 bit times, a word length of 10 bits, 25 bit times, a word length of 8 bits, 21 bit times. Additionally, with POM asserted, the SERDES will be in search mode until it finds the sync byte. In the 16-bit and 10 bit modes, the SERDES shifts out the complete sync byte and any data that follows it. In the 8 bit mode, however, the sync byte is truncated by one bit. This causes the serial data train coming from TSSDO and SDO to be one bit short. However, the data that follows the sync byte is not affected. If POM is negated and PLS is asserted (loading data into the internal register), then the serial output data will be available at the serial output ports exactly 4.5 bit times after PRDY becomes asserted. From then on, serial output data will always be 4.5 bit times delayed from PRDY. Additionally, with POM negated, the serial output port is not under SERDES control until data is loaded through the parallel port and CE is asserted. If the SERDES is left in idle mode while POM is negated, any data found at SDI0 or SDI1 will be shifted to SDO and TSSDO.
39	TSSCO H	Output	TRI STATE SERIAL CLOCK OUT—This output is a delayed version of SCI. It is delayed to match the propagation delays of TSSDO and therefore provides a positive going edge very precisely matched to the transition time of the TSSDO data bits. It is a tri state signal and is controlled by SERIAL OUTPUT TRI STATE ENABLE (SOTSE).
40	V_{CC}	—	POWER SUPPLY—This is the +5 volts supply.

Table 1-2 Word Length Select

Pin Name	16-bit	10-bit	8-bit
WLS0	H	H	L
WLS1	H	L	H

Figure 1-2 SERDES Function Block Diagram



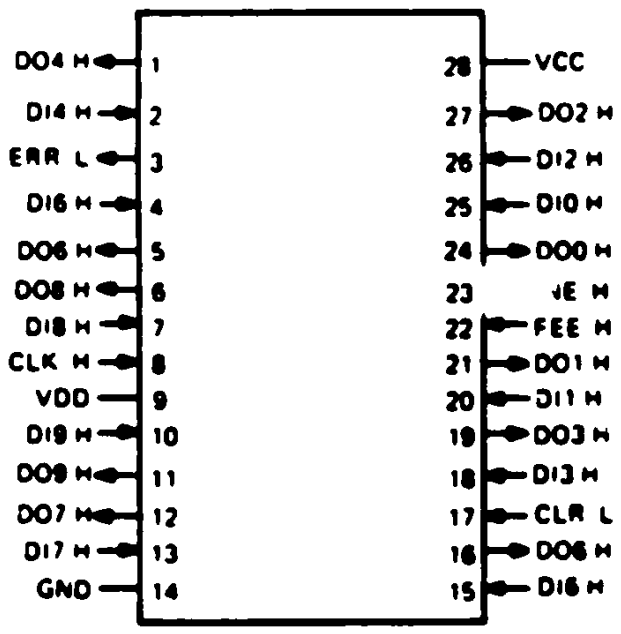
CX-1078A

1.2 Reed Solomon Generator (DC309)

The Reed Solomon Generator (R S GEN) is a dynamic N MOS LSI device that implements a Reed Solomon error correction code The chip accepts up to 10060 bits of data in 10 bit parallel (symbol) form and generates from these symbols 170 bits (17 ten-bit symbols) of error correction code (ECC) information The chip also contains hardware to compare 170 bits of calculated ECC with 170 bits of read ECC This compare is accomplished via an exclusive or operation If the data does not compare the result allows external correction of invalid data The algorithm is capable of correcting any number of errors contained in up to eight of the 10-bit symbols in the data field

Figure 1-3 shows the pin layout of the Reed Solomon Generator, and Table 1-3 describes the signal names

Figure 1-3 Reed Solomon Generator Pinout



CX-1077A

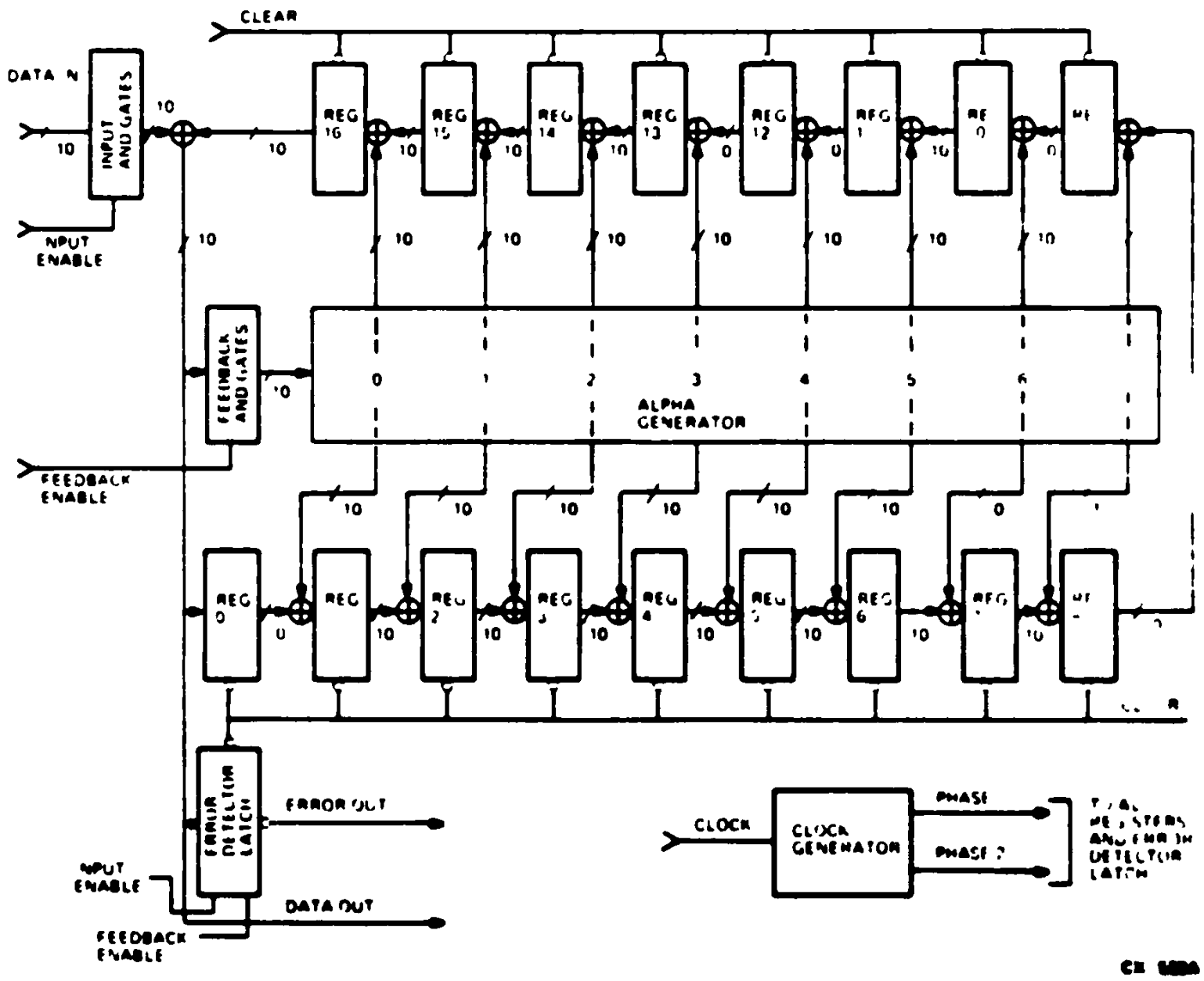
Table 1-3 Read-Solomon Generator Signal Definitions And Functions

Pin Number	Pin Name	Input Output	Signal Name Function
25	DI0 H	Input	DATA IN <9 0>—These 10 lines are the input data lines
20	DI1 H		
26	DI2 H		
28	DI3 H		
2	DI4 H		
15	DI5 H		
4	DI6 H		
13	DI7 H		
7	DI8 H		
10	DI9 H		
24	DO0 H	Output	DATA OUT <9 0>—These 10 lines are the output data lines They contain either calculated ECC or residue from an ECC comparison.
21	DO1 H		
27	DO2 H		
19	DO3 H		
1	DO4 H		
16	DO5 H		
5	DO6 H		
12	DO7 H		
67	DO8 H		
11	DO9 H		
22	FBE H	Input	FEEDBACK ENABLE—FBF is asserted when ECC is calculated It is negated when ECC is output or when ECC is compared
23	INE H	Input	INPUT ENABLE—Input enable must be asserted when the R S GEN is calculating ECC on data input and when the R S GEN is calculating the compare residue on ECC input It must be negated when the R S GEN is outputting ECC
17	CLR L	Input	CLEAR—This signal is synchronous with CLK H and is asserted for a full clock period It initializes the R S GEN
8	CLK H	Input	CLOCK—This signal provides the timing for the R S GEN During normal operation input/output signals are latched /available on the falling edge of the clock
3	ERR L	Output	ERROR OUT—This signal is negated by CLR and asserted when the ECC residue is not equal to zero indicating an ECC error The latch is enabled when FBF is negated and INE is asserted

Figure 1-4 shows the R S GEN functional block diagram The clear signal zeros the seventeen 10 bit register stages and clears the error detector The clock generator produces a two-phase clock for each of the registers The following operations are performed to generate an ECC

- The 10-bit parallel data enters through the input gate when the input enable signal is asserted
- The data passes through the feedback gate to the alpha generator when the feedback enable signal is asserted
- The alpha generator produces eight 10 bit terms composed of three to seven input bits
- The terms are exclusive OR d with the register outputs and the result clocked into the next register stage This process continues until all data words are entered

Figure 1-4 R-S GEN Functional Block Diagram

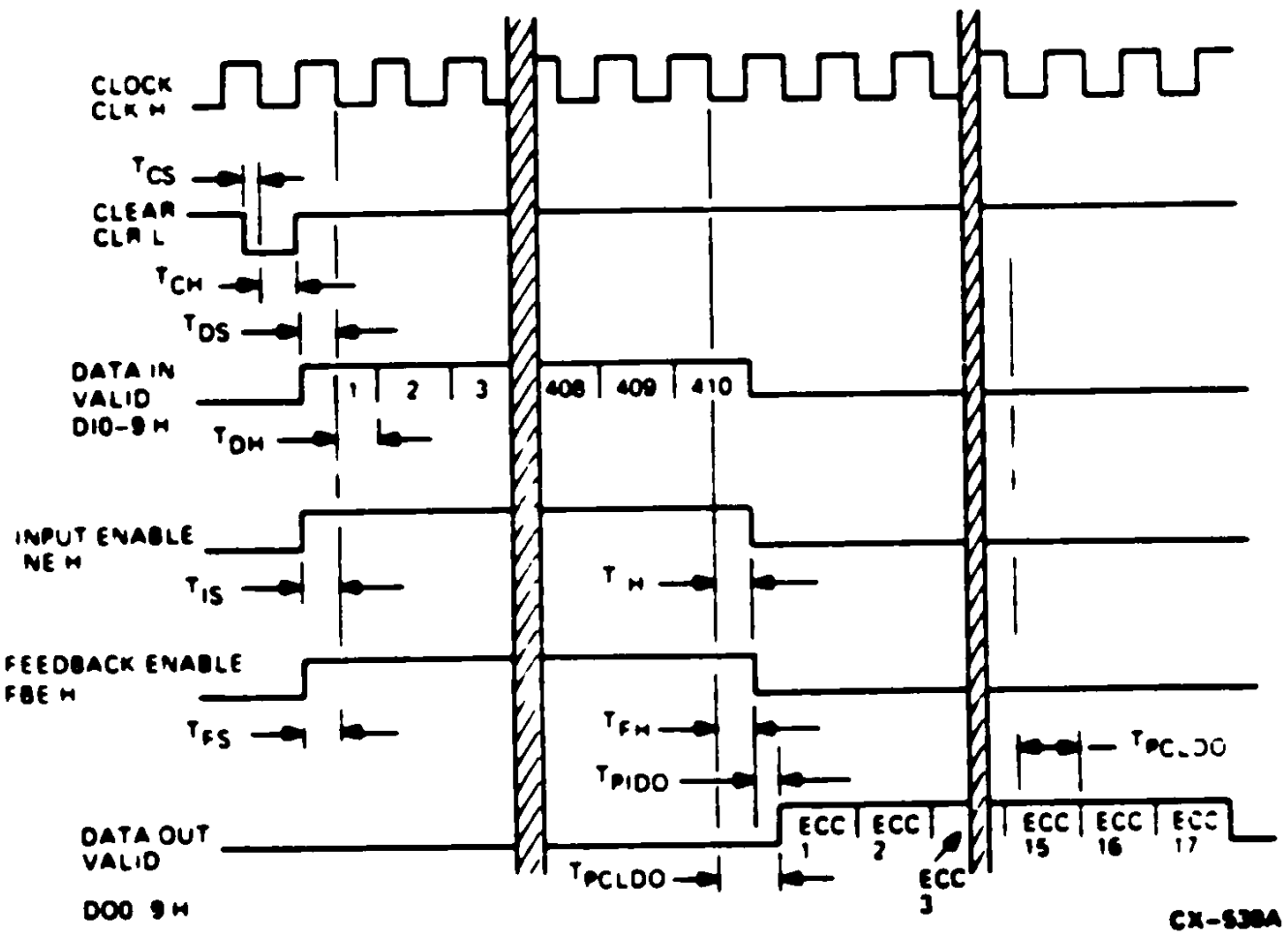


At this point the ECC is stored at the outputs of the 17 register stages To output the ECC symbols the input and feedback enables are negated and the register data shifts out at the clock frequency

To verify input data from the disk the R S GEN generates an ECC based on the input data Then the internally generated ECC is shifted through the register stages and an exclusive or operation is performed with the input ECC The result goes to the output and the error generator If the result is not zero (both ECC fields were the same) the error generator asserts the error signal

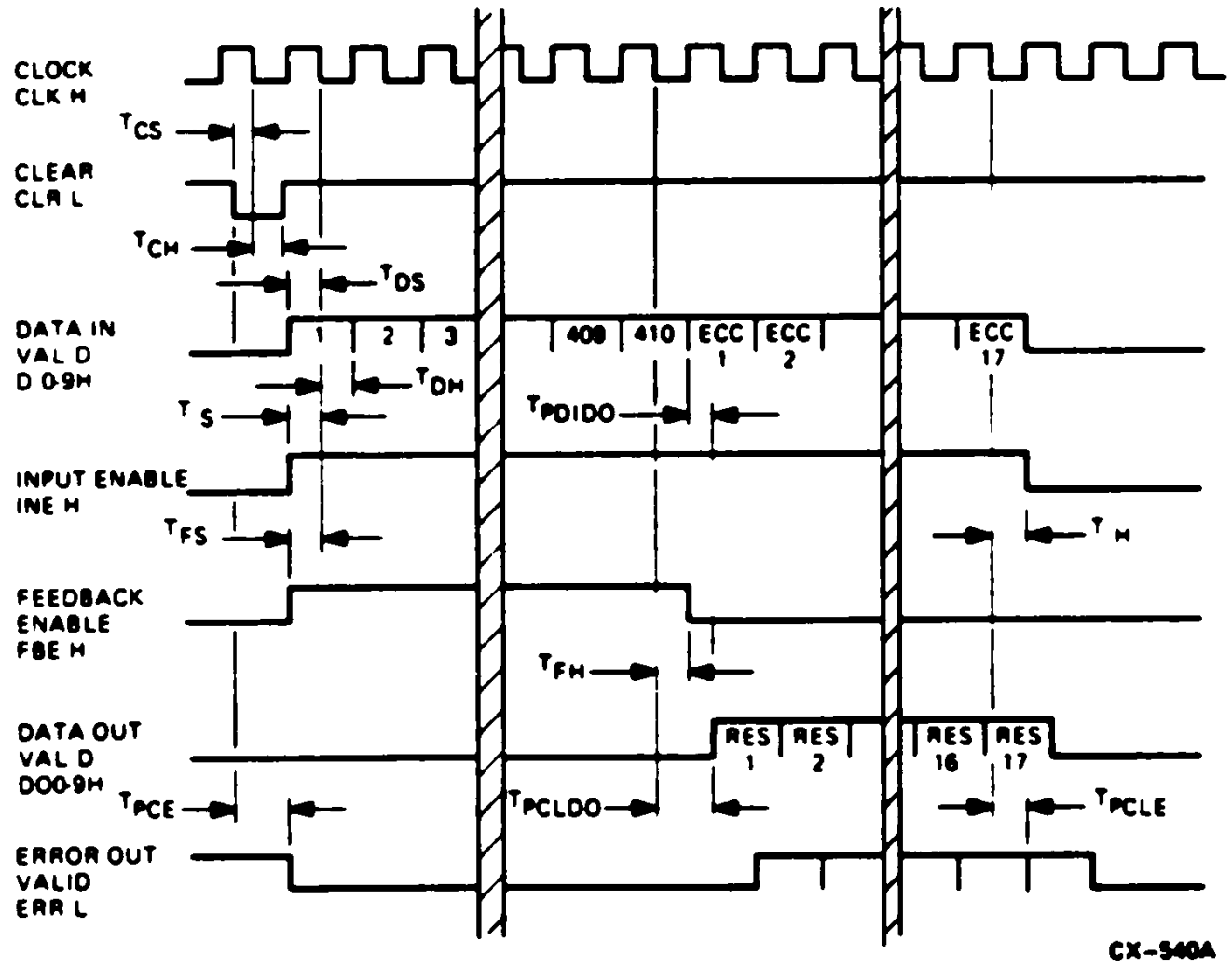
Figure 1-5 shows the timing of a typical write operation Figure 1-6 shows the timing of a typical read operation

Figure 1-5 R-S GEN Typical Write Operation



15

Figure 1-6 R-S GEN Typical Read Operation



16

1.3 REAL TIME STATE GATE ARRAY (DC703)

The Real Time State (RTS) Gate Array is a Digital Equipment Corporation-designed bipolar device for use in the Data Channel Modules. This device, in a 48 pin dual-in-line package form, provides the circuits and logic necessary to perform conversion of serial status information (and parallel information) at high data rates (minimum serial bit time = 40 nanoseconds).

The chip contains a serializer and a deserializer. The serializer converts the six asynchronous parallel status bits into a synchronous serial data stream. The data is preceded by eight zero bits (preamble) and a one (sync bit). The data bits are followed by a parity bit. The parity bit is zero when there are an odd number of ones in the six status bit, and one when there is an even number of ones in the status bits. If a parity error occurs, the parallel output buffer is not updated with the erroneous data.

The deserializer receives a serial stream of six status bits plus a parity bit and converts them to parallel outputs. The following describes the type of parallel outputs for each bit.

First bit—tri state
Second bit—tri state and totem pole
Third bit—tri state and totem-pole
Fourth bit—tri state
Fifth bit—tri-state
Sixth bit—tri-state

The deserializer looks for a sync pattern of seven or more zeros plus a one (sync bit) before it performs the serial to-parallel conversion.

Figure 1-7 shows the real time state gate array pinout. Table 1-4 provides a functional description of the RTS gate array signals.

Figure 1-7 Real Time State Gate Array Pinout

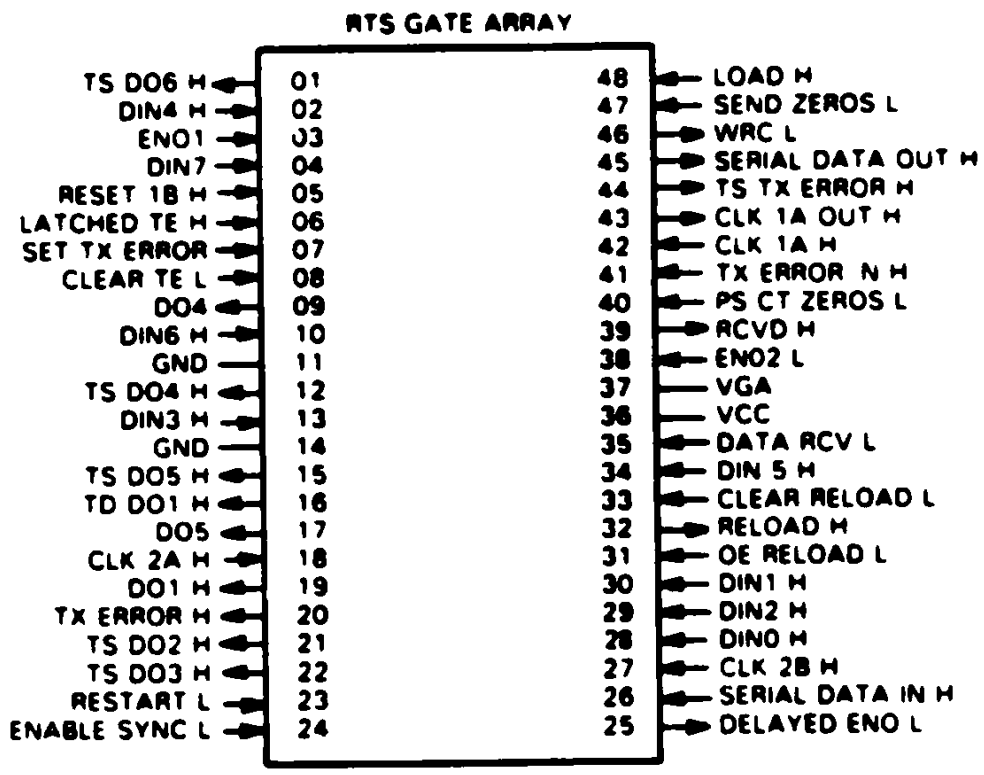


Table 1-4 RTS Gate Array Functional Signal Description

Signal Name	Description
DIN <7 0> H	Parallel data inputs to the gate array DINO H forces the gate array to send zeros on SER DATA OUT H when low, or false When high or true, it allows normal operation (See also SEND ZEROS L) DIN7 H forces a parity error
LOAD H	The rising edge of LOAD H provides a strobe signal that loads DIN <7 0> into the gate array input buffer
SEND ZEROS L	Overrides DIN0 H and forces the gate array to send zeros on SER DATA OUT H
RESET L	Causes the gate array to send zeros on SER DATA OUT H until LOAD H asserts
OE RELOAD L	The enabling signal for RELOAD H
SER DATA OUT H	Serial data output
CLK 1A H	Provides a positive edge clock signal for all of the serializer synchronous logic
CLK 1A OUT H	This is the CLK 1A H signal delayed enough to align with the SER DATA OUT H output
CLEAR RELOAD L	Clears the RELOAD H signal
RELOAD H	Indicates another parallel word can be loaded into the parallel data inputs of the gate array
WRC L	WORD RATE CLOCK—When WRC is high, the gate array is shifting out the preamble zeros When WRC is low the gate array is shifting out SYNC 1, the RTCS data, and parity
PS CT Zeros L	This input is only used for chip test
CLK2A H CLK2B H	See CLK2B Provides the positive edge triggered clock signal for all of the gate array deserializer synchronous logic CLK2A and CLK2B are normally tied together
RCVD H	Indicates a parallel data word is loaded in the output buffer
ENO1 L	Output enable for the tri-state data out lines (TS DO <6 1>)
TS DO <6 1>	Parallel data outputs for the gate array When OE1 L is asserted these outputs are enabled
DO1 DO4, DO5	Tri-state outputs for data bits 1 4 and 5
SERIAL DIN	Serial data input

Table 1-4 (Cont.) RTS Gate Array Functional Signal Description

Signal Name	Description
RESTART L	Asynchronously initializes the gate array
LATCHED TE H	Latches a transmission or a parity error until the CLEAR TE L line is asserted
TS TX Error H	This tri state signal asserts if a parity error or a transmission error occurs
TX Error H	Non tri state version of TS TX ERROR H
CLEAR TE L	Clears TX ERROR H
ENO2 L	Enables TS TX ERROR H and RCVD H
DATA RCV L	Clears RCVD H
SET TX ERROR L	Sets TX ERROR H or asserted
DELAYED ENO L	This is the output buffer load delayed by two clock times
ENABLE SYNC L	When asserted, allows the output buffer to be updated

1.4 DUAL DESERIALIZER (DC704)

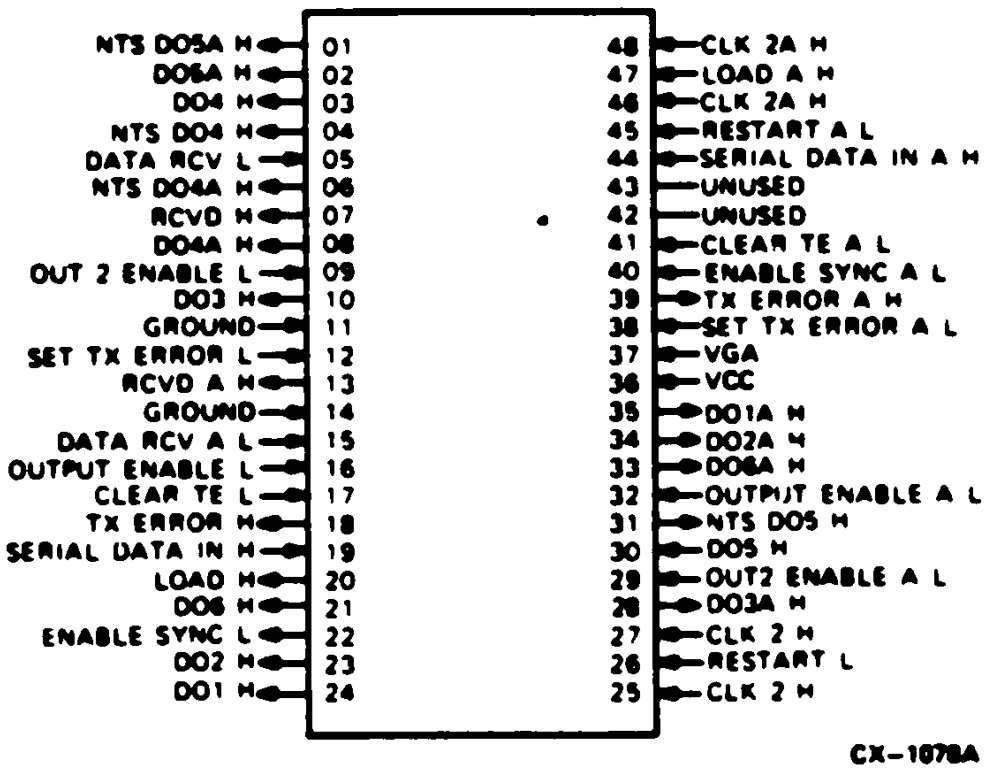
The Dual Deserializer Chip is a Digital Equipment Corporation-designed bipolar chip for use in the Disk Data Channel Module. This chip provides the circuits and logic necessary to perform conversion of serial status information at high data rates (minimum serial bit time is 40 nanoseconds).

This chip contains two deserializers. The deserializers receive serial stream of six status bits plus a parity bit and converts these to parallel outputs. All six outputs are tri-state with bit four and five being TTL totem pole outputs as well. The number of ones in the six status bits plus the parity bit must always be odd. If a parity error occurs, the parallel output buffer will not be updated with erroneous data.

The deserializer looks for a sync pattern of eight or more zeros plus a one sync bit before it performs the serial-to-parallel conversion.

Figure 1-8 shows the pin layout of the dual deserializer and Table 1-5 provides a functional description of the signals on the output pins. Figure 1-9 is a functional block diagram of the dual deserializer.

Figure 1-8 Dual Deserializer Pin Layout



CX-1078A

Table 1-5 Dual Deserializer Signal Definitions and Functions

Pin Number	Function	Description
1	NTS DO5A H	The non tri state (TTL) output bit five
2	DATA OUT 5A H	Parallel output data for the D5A data bit Outputs are tri-state and are enabled when OUTPUT ENABLE A L is asserted
3	DATA OUT 4 H	Parallel output data for the D4 data bit Outputs are tri-state and are enabled when OUTPUT ENABLE L is asserted
4	NTS DO4 H	The non tri state (TTL) output bit four
5	DATA RCV L	This input clears the RCVD H output latch
6	NTS DO4A H	The non tri state (TTL) output bit four
7	RCVD H	This TTL output indicates that a frame has just been loaded to the output buffer
8	DATA OUT 4AS H	Parallel output data for the D4A data bit Outputs are tri state and are enabled when OUTPUT ENABLE A L is asserted
9	OUTPUT 2 ENABLE L	The nput which enables tri-state outputs—RCVD H and TX ERROR H
10	DATA OUT 3 H	Parallel output data for the D3 data bit Outputs are tri state and are enabled when OUTPUT ENABLE L is asserted
11	GND	Ground
12	SET TX ERROR L	This input sets the TX ERROR flip flop to a high
13	RCVD A H	This TTL output indicates that a frame has just been loaded to the output buffer
14	GND	Ground
15	DATA RCV A L	This input clears the RCVD A H output latch
16	OUTPUT ENABLE L	The input which enables the tri state outputs (DATA OUT 1 H DATA OUT 2 H DATA OUT 3 H DATA OUT 4 H DATA OUT 5 H and DATA OUT 6 H)
17	CLEAR TE L	Clear out the latched TX ERROR signal
18	TX ERROR H	A TTL output which indicates that a parity or transmission error has occurred
19	SERIAL DATA IN H	Serial date in
20	LOAD H	A synchronizing signal that loads data from the output buffer to the final output latch

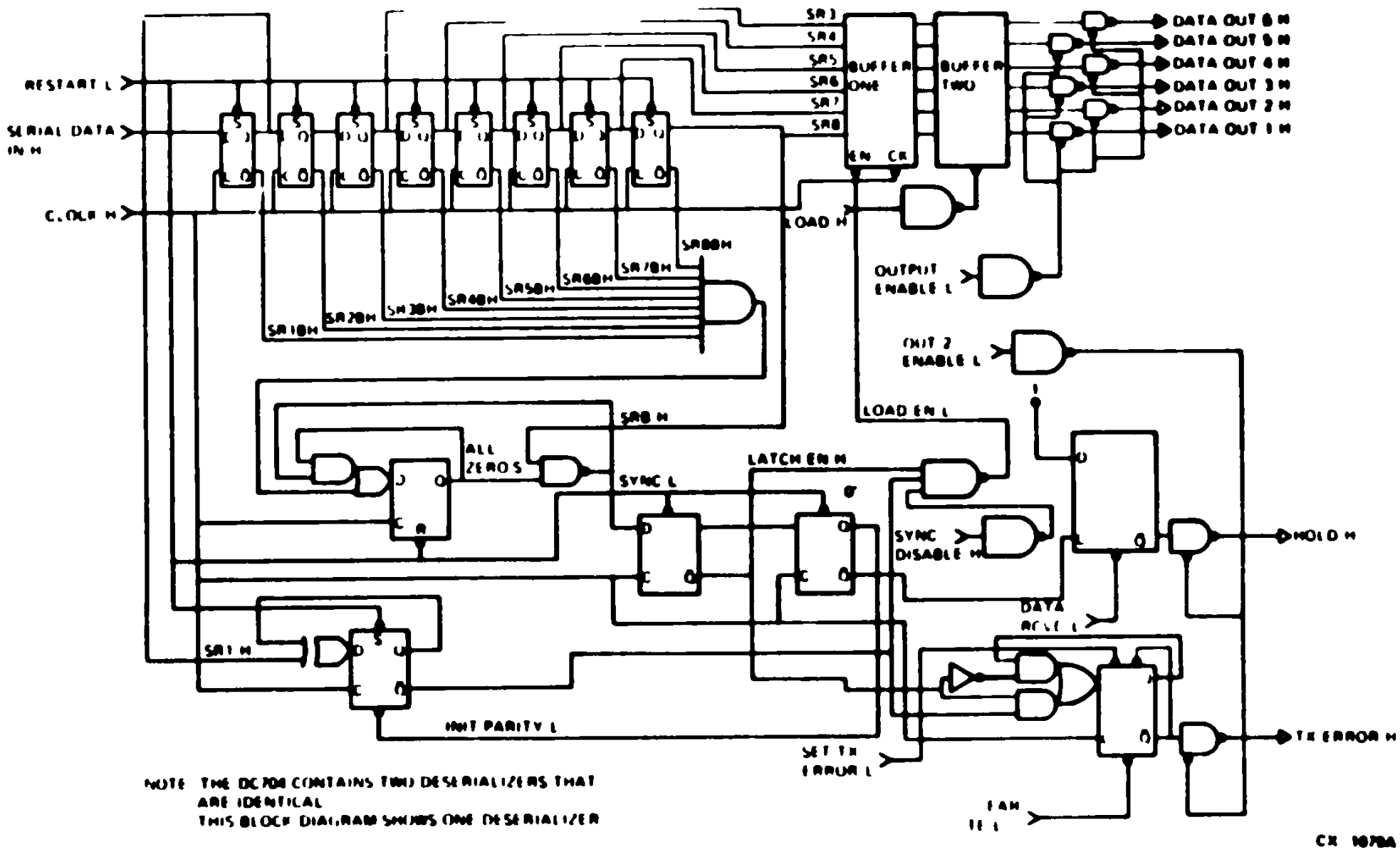
Table 1-5 (Cont.) Dual Deserializer Signal Definitions and Functions

Pin Number	Function	Description
21	DATA OUT 6 H	Parallel output data for the D6 data bit. Outputs are tri state and are enabled when OUTPUT ENABLE L is asserted
22	ENABLE SYNC L	When asserted the output buffer cannot be updated
23	DATA OUT 2 H	Parallel output data for the D2 data bit. Outputs are tri state and are enabled when OUTPUT ENABLE L is asserted
24	DATA OUT 1 H	Parallel output data for the D1 data bit. Outputs are tri state and are enabled when OUTPUT ENABLE L is asserted
25	CLK2 H	The clock signal for all the deserializer synchronous logic. The logic is positive edge triggered. This pin is tied to pin 27
26	RESTART L	Asynchronously sets the chip to it initial conditions
27	CLK 2 H	The clock signal for all the deserializer synchronous logic. The logic is positive edge triggered. This pin is tied to pin 25
28	DATA OUT 3A H	Parallel output data for the D3A data bit. Outputs are tri state and are enabled when OUTPUT ENABLE A L is asserted
29	OUTPUT 2 ENABLE A L	The input which enables tri state outputs—RCVD H and TX ERROR H
30	DATA OUT 5 H	Parallel output data for the D5 data bit. Outputs are tri state and are enabled when OUTPUT ENABLE L is asserted
31	NTS DO5 H	The non tri state (TTL) output bit five
32	OUTPUT ENABLE A L	The input which enables the tri state outputs (DATA OUT 1A H DATA OUT 2A H DATA OUT 3A H DATA OUT 4A H DATA OUT 5A H and DATA OUT 6A H)
33	DATA OUT 6A H	Parallel output data for the D6A data bit. Outputs are tri state and are enabled when OUTPUT ENABLE A L is asserted
34	DATA OUT 2A H	Parallel output data for the D2A data bit. Outputs are tri state and are enabled when OUTPUT ENABLE A L is asserted
35	DATA OUT 1A H	Parallel output data for the D1A data bit. Outputs are tri state and are enabled when OUTPUT ENABLE A L is asserted
36	V_{CC}	+5.0 volts
37	V_{CA}	Supplied from V_{CC} through a 10 ohm ± 5% 2 watt metal film resistor
38	SET TX ERROR A H	This input sets the TX ERROR flip flop to a high
39	TX ERROR A H	A H output that indicates a parity or transmission error has occurred

Table 1-5 (Cont.) Dual Deserializer Signal Definitions and Functions

Pin Number	Function	Description
40	ENABLE SYNC A L	When asserted the output buffer cannot be updated
41	CLEAR TE A L	Clear out the latched TX ERROR signal
42	Unused	
43		
44	SERIAL DATA IN A H	Serial data in
45	RESTART A L	Asynchronously sets the chip to its initial buffer
46	CLK 2A H	The clock signal for all the deserializer synchronous logic Positive edge triggered This pin is tied to pin 48
47	LOAD A H	A synchronizing signal that loads data from the output buffer to the final output latch
48	CLK 2A H	The clock signal for all the deserializer synchronous logic Positive edge triggered This pin is tied to pin 46

Figure 1-9 Dual Deserializer Functional Block Diagram

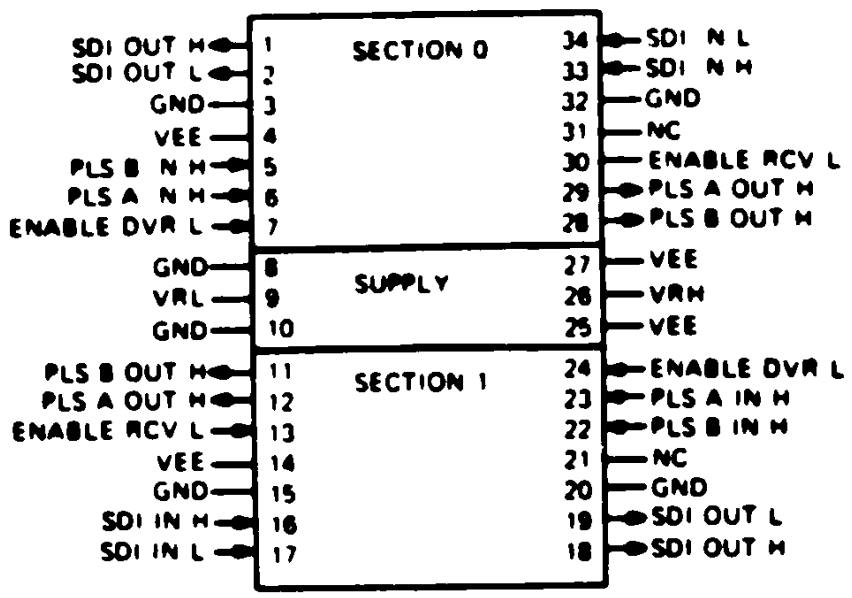


1.5 SDI STI RECEIVER DRIVER (DC026)

The SDI/STI Receiver/Driver chip is a Digital Equipment Corporation designed hybrid chip. This chip provides the circuit and logic necessary to perform the receiving and driving function for the SDI/STI bus at high data rates (minimum serial bit time is 40 nanoseconds). The driver receives 12 nanosecond pulses on the PLSA and PLSB inputs which have previously been encoded by the SDI/STI Encoder /Decoder chip. The driver (designed for a 93 ohm coax) converts these pulses into a three level signal and drives the SDI/STI cable. The receiver circuitry receives the three level signal from the cable and converts it into pulses on the PLSA and PLSB outputs. The signals are then decoded into NRZ data and clocked by the Encoder/Decoder hybrid chip. This chip uses ECL signal levels.

Figure 1-10 shows the pin layout of the dual deserializer and Table 1-6 provides a functional description of the signals on the output pins. Figure 1-11 is a functional block diagram of the dual deserializer.

Figure 1-10 SDI/STI Receiver Driver Pin Layout



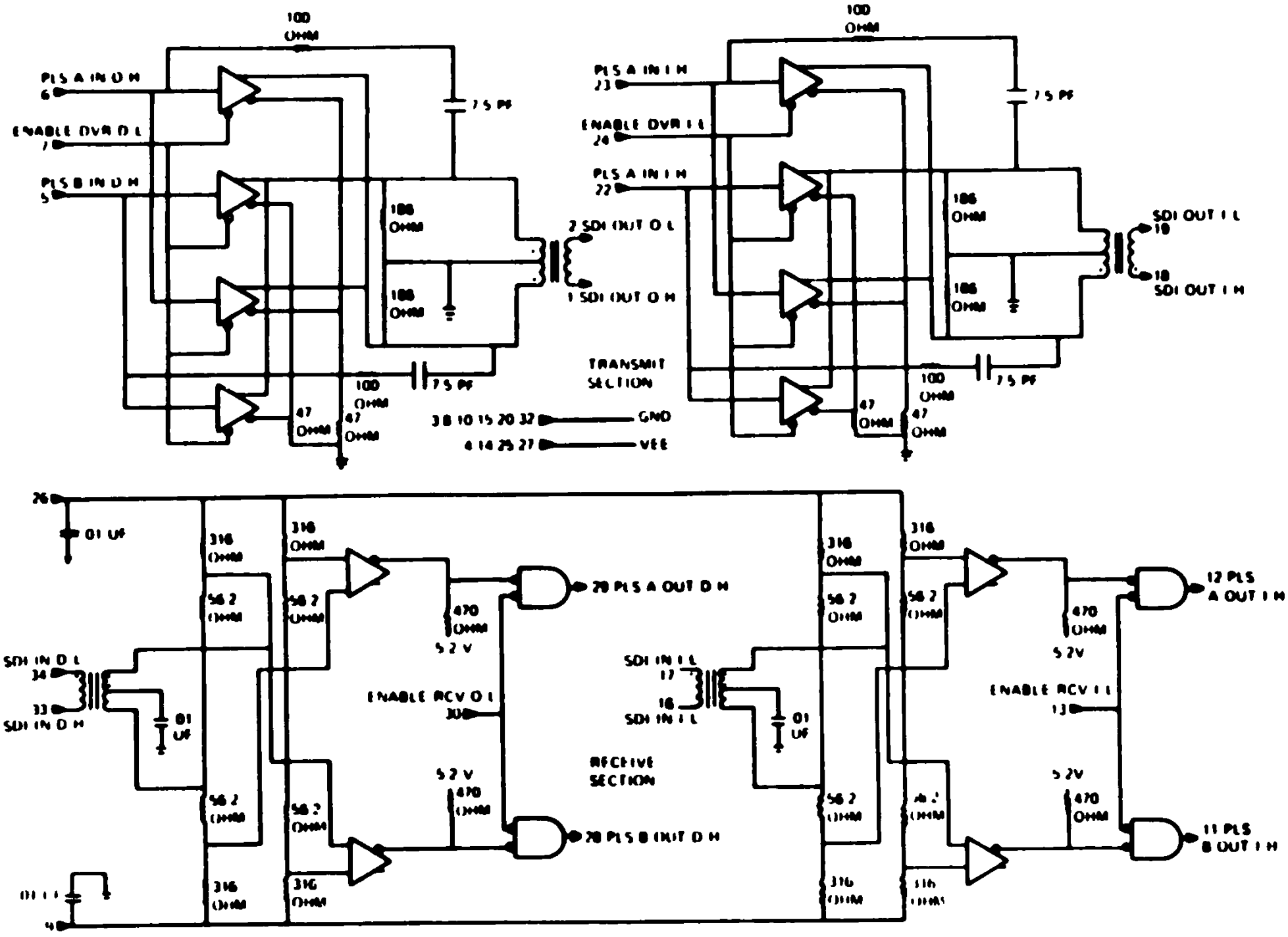
CX-1681A

25

Table 1-6 SDI STI Receiver Driver Signal Definitions and Functions

Pin Numbers	Description	Function
6 23	PLS A IN H	Input signal from the encoder PULSE A output A high pulse on this input causes the signal at SDI OUT to pulse above the zero volt level
5 22	PULSE B IN H	Input signal from the encoder PULSE B output A high pulse on this input causes the signal at SDI OUT to pulse below the zero volt level
7 24	ENABLE DVR L	Enable input for the SDI OUT H and SDI OUT L outputs maintains a zero volt level at the SDI outputs when HIGH
2 19	SDI OUT L	The negative output connection to the SDI cable
1 18	SDI OUT H	The positive output connection to the SDI cable
3 8 10 15 20 32	GND	Pins 8 and 10 are major current path, pins 3 15 20, and 32 are additional ground paths
4 14 25 27	V_{EE}	The -5 2 volt supply pins
17 34	SDI IN L	The negative input connection from the SDI cable
16 33	SDI IN H	The positive input connection from the SDI cable
13 30	ENABLE RCV L	Enables the receive section when low
12 29	PLS A OUT H	The output which pulses high when the signal at SDI pulses above the zero volt level It drives PULSE A IN H of the decoder
11 28	PLS B OUT H	The output which pulses high when the signal at SDI pulses below the zero volt level It drives PULSE B IN H of the decoder
26	VRH	Receive reference voltage high (+0 6 volts)
9	VRI	Receive reference voltage low (-3 3 volts)
21 31		No connection

Figure 1-11 SDI STI Receiver Driver Functional Block Diagram



CX 1888A

27

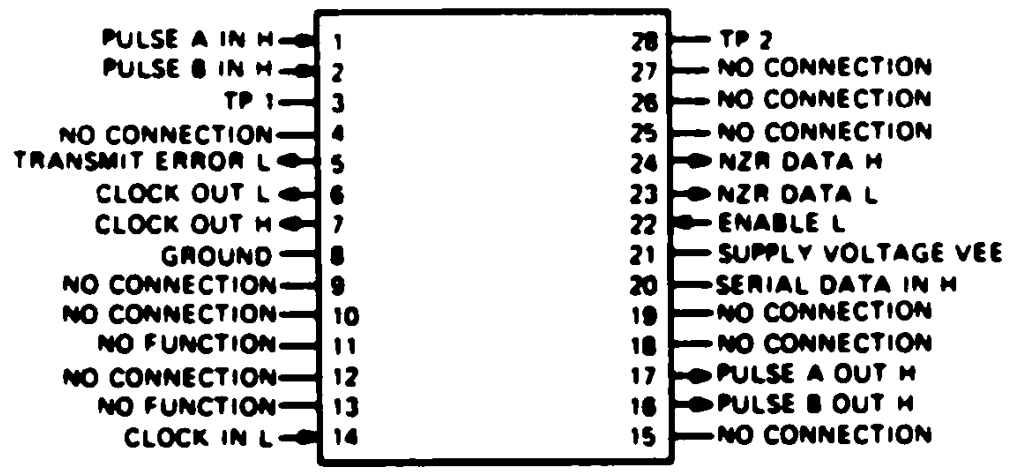
1.6 SDI/STI ENCODER DECODER (DC027)

The SDI/STI Encoder/Decoder chip is a Digital Equipment Corporation designed hybrid chip This chip performs the encoding and decoding function for the SDI/STI bus

The chip contains an encoder and a decoder The encoder receives serial NRZ data and clock as inputs and utilizes a self clocked encoding scheme to translate them into a series of 12 nanosecond pulses on the pulse A and pulse B outputs These outputs drive the SDI/STI Receiver/Driver chips which drive the SDI/STI cable with the encoded signal The decoder receives a series of pulses from the Receiver/Driver chip and performs a decoding algorithm to regenerate NRZ data and clock This chips uses ECL signal levels

Figure 1-12 shows the pin layout of the dual deserializer and Table 1-7 provides a functional description of the signals on the output pins Figure 1-13 is a functional block diagram of the dual deserializer

Figure 1-12 SDI/STI Encoder/Decoder Pin Layout



CX-1082A

Table 1-7 SDI STI Encoder Decoder Signal Definitions and Functions

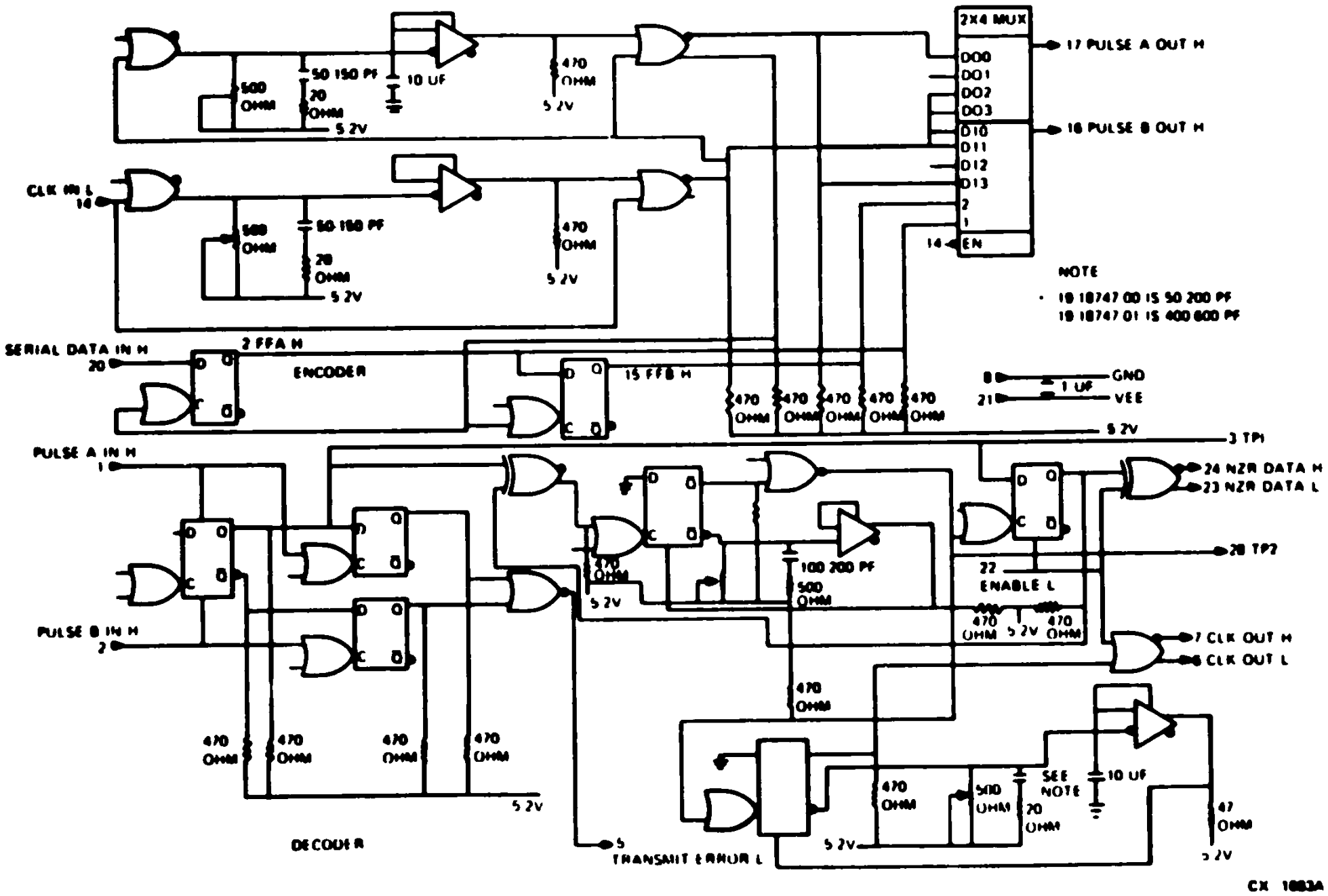
Pin Number	Description	Function
1	PULSE A IN H	One of the pulses (approximately 12 nanoseconds) supplied to the decoder This signal, in conjunction with PULSE B IN H, determines the NRZ data pattern and clock This signal is supplied from pin 29 (PLS A OUT 0 H) or pin 12 (PLS A OUT 1 H) of the SDI/STI RCV/DVR hybrid chip
2	PULSE B IN H	One of the pulses (approximately 12 nanoseconds) supplied to the decoder This signal, in conjunction with PULSE A IN H, determines the NRZ data pattern and clock This signal is supplied from pin 28 (PLS B OUT 0 H) or pin 11 (PLS B OUT 1 H) of the SDI/STI RCV/DVR hybrid chip
3	TP1	No function Used during manufacture for resistor trimming of the decoder
4		No connection
5	TRANSMIT ERROR L	This output will pulse low whenever a transmission error has occurred (For example, extra pulses or a missed pulse on either PULSE A or PULSE B inputs to the decoder)
6 7	CLK OUT L CLK OUT H	The decoded clock signal of the decoder synchronous to NRZ DATA H and NRZ DATA L
8	GND	Ground
9		No connection
10		
11		No function, internal connection
12		No connection
13		No function, internal connection
14	CLOCK IN L	The clock input signal for the encoder, negative edge triggered This is the timing reference for PULSE A OUT H and PULSE B OUT H
15		No connection
16	PULSE B OUT H	One of the 12-nanosecond pulses out of the encoder This signal is connected to pin 5 (PLS B IN 0 H) or pin 27 (PLS B IN 1 H) of the SDI/STI RCV/DVR hybrid chip
17	PULSE A OUT H	One of the 12 nanosecond pulses out of the encoder The absence or presence and position of this pulse with respect to PULSE B OUT H determines the encoded data pattern This signal is connected to pin 6 (PLS A IN 0 H) or pin 23 (PLS A IN 1 H) of the SDI STI RCV/DVR hybrid chip

Table 1-7 (Cont) SDI STI Encoder Decoder Signal Definitions and Functions

Pin Number	Description	Function
18		No connection
19		
20	SERIAL DATA IN H	The NRZ serial data supplied to the encoder
21	V_{EE}	-5 2 volts
22	ENABLE L	A high at this decoder input will cause
		CLOCK OUT L (pin 6) to be in the high state
		CLOCK OUT H (pin 7) to be in the low state
		NRZ DATA H (pin 24) to be in the low state
		NRZ DATA L (pin 23) to be in the high state
		As long as this input is high, the PULSE A and PULSE B inputs have no effect on the output When this input goes low the clock and NRZ DATA are controlled by the clock PULSE A and PULSE inputs
23	NRZ DATA L	The decoded NRZ data outputs which are synchronous with CLK OUT H and CLK OUT L
24	NRZ DATA H	
25		No connection
26		
27		
28	TP2	No function used during manufacture for active trimming

30

Figure 1-13 SDI/STI Encoder/Decoder Functional Block Diagram



CHAPTER 2
HSC50/70 PAL EQUATIONS

This chapter contains the PAL equations for the PALs used in the HSC50 and HSC70 This chapter is arranged by module in the following order

HSC50 Input/Output Control Processor Module (P ioc)
HSC70 Input/Output Control Processor Module (P ioj)
HSC70 Memory Module
Data Channel Module (K sdi and K sti)

2.1 PAL Equations for PALs on the HSC50 Input/Output Control Processor Module

Figures 2-1 through 2-5 are the PAL equations for the PALs on the HSC50 Input/Output Control Processor Module

2.2 PAL Equations for PALS on the HSC70 Input/Output Control Processor Module

Figures 2-6 through 2-13 are the PAL equations for the PALs on the HSC70 Input/Output Control Processor Module

2.3 PAL Equations for PALS on the HSC70 Memory Module

Figures 2-14 through 2-22 are the PAL equations for the PALs on the HSC70 Memory Module

2.4 PAL Equations for PALS on the Data Channel Module

Figure 2-23 is the PAL equation for the PAL on the Data Channel Module

Figure 2-1 23-062J5-00

```
DATE   3-APR 80
----
SYMBOL TABLE
------------
                    --- -- --
              01 = 1 *   2   * 20 = VCC
              02 = 2 *   3   * 19 = /19
              03 = 3 *       * 18 = /P18_RD_WADR_REG
              04 = 4 *   0   * 17 = /P18_RD_WIND_REG
              05 = 5 *   6   * 16 = /P18_RD_WBUS_REG
              06 = 6 *   2   * 15 = /P18_RD_REPLY_0
              07 = 7 *   J   * 14 = /P18_RD_LO_EAR
              08 = 8 *   5   * 13 = /P18_RD_HI_EAR
              09 = 9 *   2   * 12 = 12
              10 =10 *   2   * 11 = 11
                    ---------

EQUATIONS
---------
IF [VCC] /19 = 01 * /02 * /03 * /04 * /05 * 09
.ENABLE DECODER FOR ADDRESSES 17 770 040 - 056

IF [VCC] /18_RD_WADR_REG = 01 * /02 * /03 * /04 * /05 * 09
.ENABLE ADDRESSES 17 770 000 - 016

IF [VCC] /18_RD_WIND_REG = 01 * /02 * /03 * /04 * 05 * /06
* /07 * /08 * 09
.DECODES 17 770 020

IF [VCC] /18_RD_WBUS_REG = 01 * /02 * /03 * /04 * 05 * /06
* /07 * 08 * 09
;DECODES 17 770 022

IF [VCC] /18_RD_REPLY_0 = 01 * /02 * /03 * 04 * 05 * 09
* 01 * /02 * /03 * /04 * /05 * 09
* 01 * /02 * /03 * /04 * 05 * /06 * /07 * /08 * 09
* 01 * /02 * /03 * /04 * 05 * /06 * /07 * 08 * 09
* 01 * /02 * /03 * /04 * 05 * /06 * 07 * /08 * 09
* 01 * /02 * /03 * /04 * 05 * /06 * 07 * 08 * 09
;INCLUSIVE OR OF PREVIOUS FOUR TERMS AND THE FOLLOWING TWO TERMS

IF [VCC] /P18_RD_LO_EAR = 01 * /02 * /03 * /04 * 05 * /06 * 07 * /08 * 09
;DECODES 17 770 024

IF [VCC] /P18_RD_HI_EAR = 01 * /02 * /03 * /04 * 05 * /06 * 07 * 08 * 09
;DECODES 17 770 026

TITLE  E93 IO PAGE RD DECODER (LOW) 16L8
NOTE 062J5  REV A     23 062J5 01 (PIOC1)
```

Figure 2-2 23-063J5-00

```
DATE:  3-APR-80
-----
SYMBOL TABLE
------------
                    ---------
              01 = 1 *   2   * 20 = VCC
              02 = 2 *   3   * 19 = /P18_RD_RCSR
              03 = 3 *       * 18 = /P18_RD_RBUF
              04 = 4 *   0   * 17 = /P18_RD_XCSR
              05 = 5 *   6   * 16 = /P18_DL_SEL_1
              06 = 6 *   2   * 15 = /P18_DL_SEL_0
              07 = 7 *   J   * 14 = /P18_RD_REPLY_1
              08 = 8 *   5   * 13 = 13
              09 = 9 *   2   * 12 = 12
              10 =10 *   2   * 11 = 11
                    ---------

EQUATIONS
---------
IF [VCC] /P18_RD_RCSR = /01 * 02 * /03 * 04 * /05 * 06 * /08
* /09 * /11
* /01 * 02 * /03 * 04 * 05 * 06 * /07 * /08 * /09 * /11
;ENABLES ADDRESSES 17 777 520, -530, AND -560

IF [VCC] /P18_RD_RBUF = /01 * 02 * /03 * 04 * /05 * 06 * /08
* 09 * /11
* /01 * 02 * /03 * 04 * 05 * 06 * /07 * /08 * 09 * /11
;ENABLES ADDRESSES 17 777 522, -532, AND -562

IF [VCC] /P18_RD_XCSR = /01 * 02 * /03 * 04 * /05 * 06 * 08
* /09 * /11
* /01 * 02 * /03 * 04 * 05 * 06 * /07 * 08 * /09 * /11
,ENABLES ADDRESSES 17 777 524, -534, AND -564

IF [VCC] /P18_DL_SEL_1 = /01 * 02 * /03 * 04 * 06 * /07 * /11
,ASSERTED FOR TU-1 AND TERM REGISTERS

IF [VCC] /P18_DL_SEL_0 = /01 * 02 * /03 * 04 * /05 * 06 * /07
* /11
* /01 * 02 * /03 * 04 * 05 * 06 * /07 * /11
,ASSERTED FOR TU-2 AND TERM REGISTERS

IF [VCC] /P18_RD_REPLY_1 = /01 * 02 * /03 * 04 * /05 * 06 * /11
* /01 * 02 * /03 * 04 * 05 * 06 * /07 * /11

TITLE  E94 IO PAGE RD DECODER (HIGH) 16L8
NOTE  PAT-063J5     REV A   23-063J5-1 (PIOC2)
```

Figure 2-3 23-064J5-00

```
DATE   3-APR-80
-----
SYMBOL TABLE
------------
                    ---------
              01 = 1 *   2   * 20 = VCC
              02 = 2 *   3   * 19 = /P18_LD_WADER_REG
              03 = 3 *       * 18 = /P18_LD_WIND_REG
              04 = 4 *   0   * 17 = /P18_LD_PIO_CSR
              05 = 5 *   6   * 16 = /P18_LD_SWD_REG
              06 = 6 *   2   * 15 = /P18_LD_KIN_REG
              07 = 7 *   J   * 14 = /P18_LD_KST_REG
              08 = 8 *   5   * 13 = /P18_LD_REPLY_0
              09 = 9 *   2   * 12 = 12
              10 =10 *   2   * 11 = 11
                    ---------
EQUATIONS
---------
IF [VCC] /P18_LD_WADR_REG = 01 * /02 * /03 * /04 * /05 * 09
.ENABLES ADDRESSES 17 777 000 - 016

IF [VCC] /P18_LD_WIND_REG = 01 * /02 * /03 * /04 * /05 * /06 * /07
* 08 * 09
.ADDRESS 17 777 020

IF [VCC] /P18_LD_PIO_CSR = 01 * /02 * /03 * 04 * /05 * /06 * /07
* /08 * 09
.ADDRESS 17 777 040

IF [VCC] /P18_LD_SWD_REG = 01 * /02 * /03 * 04 * /05 * /06 * /07
* 08 * 09
.ADDRESS 17 777 042

IF [VCC] /P18 LD KIN REG = 01 * /02 * /03 * 04 * /05 * /06 * 07
* /08 * 09
.ADDRESS 17 777 044

IF [VCC] /P18_LD KST REG = 01 * /02 * /03 * 04 * /05 * /06 * 07
* 08 * 09
.ADDRESS 17 777 026

IF [VCC] /P18_LD REPLY_0 = 01 * /02 * /03 * /04 * /05 * 09
* 01 * /02 * /03 * /04 * 05 * /06 * /07 * /08 * 09
* 01 * /02 * /03 * 04 * /05 * /06 * 09

TITLE  E103 IO PAGE LD DECODER (LOW) 16L8
NOTE  PAT-064J5 REV A 23 064J5 01 (PIOC3)
```

Figure 2-4 23-065J5-00

```
DATE   3-APR-00
-----
SYMBOL TABLE
------------
                    ------- -
              01 = 1 *   2   * 20 = VCC
              02 = 2 *   3   * 19 = /19
              03 = 3 *       * 18 = /P18_LD_TERM RCSR
              04 = 4 *   0   * 17 = /P18 LD_TERM XCSR
              05 = 5 *   6   * 16 = /P18 LD TERM XBUF
              06 = 6 *   2   * 15 = /P18_LD REPLY_1
              07 = 7 *   J   * 14 = 14
              08 = 8 *   5   * 13 = 13
              09 = 9 *   2   * 12 = 12
              10 =10 *   2   * 11 = 11
                    ---  -
EQUATIONS
---------
IF [VCC] /19 = /01 * 02 * /03 * 04 * /05 * 06 * /11
.ENABLES ADDRESSES 17 777 520 - 536

IF [VCC] /P18_LD_TERM RCSR = /01 * 02 * /03 * 04 * 05 * 06
* /07 * /08 * /09 * /11
.ADDRESS 17 777 560

IF [VCC] /P18_LD_TERM_XCSR = /01 * 02 * /03 * 04 * 05 * 06
* /07 * 08 * /09 * /11
.ADDRESS 17 777 564

IF [VCC] /P18_LD_TERM_XBUF = /01 * 02 * /03 * 04 * 05 * 06
* /07 * 08 * 09 * /11
.ADDRESS 17 777 566

IF [VCC] /P18_LD_REPLY_1 = /01 * 02 * /03 * 04 * /05 * /11
* /01 * 02 * /03 * 04 * 05 * 06 * /07 * /11
.

TITLE  E113 IO PAGE LD DECODER (HIGH) 16L8
NOTE  PAT-065J5   REV A  23 065J5 01 (PIOC4)

1) REPLY IS GENERATED IF AN ATTEMPT TO WRITE AN RBUF IS MADE
HOWEVER. THE DATA IS DUMPED ON THE FLOOR (RBUFS ARE
READ-ONLY)
```


3.2

Figure 2-5 23-135K5-00

```
T    PAL16R8
P    23 135K5 00
N    HARDWARE ENGINEER
D    22 OCT 1985
S    01 02 03 04 05 06 07 08 09  GND /11 /CGRANT 7 /CGRANT 6 /CGRANT 5
     /CGRANT 4 /CGRANT 3 /CGRANT 2 /CGRANT 1 /CGRANT 0 VCC
B    /CGRANT 0 = 02 • /03 • /04 • /05 • /06 • /07 • /08 • /09
     /CGRANT 1 = 03 • /04 • /05 • /06 • /07 • /08 • /09
     /CGRANT 2 = 04 • /05 • /06 • /07 • /08 • /09
     /CGRANT 3 = 05 • /06 • /07 • /08 • /09
     /CGRANT 4 = 06 • /07 • /08 • /09
     /CGRANT 5 = 07 • /08 • /09
     /CGRANT_6 = 08 • /09
     /CGRANT_7 = 09
E This PAL generates CONTROL BUS GRANTS in response to CONTROL BUS
REQUESTS on the L0106 (P iec) module of the HSC50   REQ7 is the highest
priority and REQ0 is the lowest priority
The base part number for this PAL is 23-000K5-04
```

Figure 2-6 23-161K3-00

```
T    PAL16R4
P    23-161K3-00
N    HARDWARE ENGINEER
D    27-OCT-84

S    STB (2) (3) 4 5 6 7 8 9 GND
       (ENABLE) 12 13 /R3 /R2 /R1 /R0 18 19 VCC

B    /R0  = 12 * /6 * /8 * /4 * /13 * /19
            + 7 * /8 * /4 * /13 * /19
            + 9 * /4 * /13 * /19
            + 5 * /13 * /19
            + 18 * /19

       /R1  = 12 * /6 * /7 * /9 * /18 * /19
            + 8 * /9 * /18 * /19
            + 4 * /18 * /19
            + 5 * /18 * /19
            + 13 * /18 * /19

       /R2  = 12 * /9 * /4 * /5
            + 6 * /9 * /4 * /5
            + 7 * /9 * /4 * /5
            + 8 * /9 * /4 * /5
            + 13
            + 18
            + 19

       /R3  = 6 * /13 * /18 * /19
            + 7 * /13 * /18 * /19
            + 8 * /13 * /18 * /19
            + 9 * /13 * /18 * /19
            + 4 * /13 * /18 * /19
            + 5 * /13 * /18 * /19
```

(Continued on next page)

Figure 2-6 (Cont) 23-161K3-00

E END OF EQUATIONS

E133. INTERRUPT PRIORITY ENCODER

This PAL generates a 4-bit code corresponding to the highest priority active interrupt of the ten possible interrupts on the module Synchronization of the interrupts is provided external to the PAL

Valid codes are defined in the following table

	R3	R2	R1	R0
CONTROL BUS 7	H	L	H	H
CONTROL BUS 6	H	L	H	L
CONTROL BUS 5	H	L	L	H
TERMINAL RCVR	L	H	L	L
TERMINAL XMTR	L	H	L	H
AUX1 RCVR	L	H	H	L
AUX1 XMTR	L	L	L	H
AUX2 RCVR	L	L	H	L
AUX2 XMTR	L	L	H	H
CONTROL BUS 4	H	L	L	L
NO INTERRUPTS ACTIVE	H	H	H	H

NOTE Input terms (2) and (3) are not presently used, but could be used for two additional interrupt inputs Interrupts are encoded asynchronously and appear at the registered outputs (R3 0) after the clock occurs

This PAL is used on the LO111 (P ,ej) module of the HSC70

The base part number for this PAL is 23-000K3-03

Figure 2-7 23-258J5-00

```
T    PAL16L8
P    23-258J5-00
N    HARDWARE ENGINEER
D    23-MAR-83

S    01 02 03 04 05 06 07 08 09 GND
     11 12 13 14 15 16 17 18 19 VCC

B    IF [VCC] /17 = 03 * 04 * 05 * 06 * 07 * 08 * 09 * 11 * 13 * 14
     + 01
     + /02 * 03

     IF [VCC] /16 = /01 * /03 * 04 * 05 * 06 * 07 * 08 * 09 * 11 * 13 * 14

     IF [VCC] /15 = /01 * 02 * /04 * 05 * 06 * 07 * 08 * 09 * 11 * 13 * 14

E    END OF EQUATIONS

TITLE   Low Order Bus Arbitrater

This PAL is used for both the CONTROL BUS and the DATA BUS ARBITRATORS
in conjunction with 23-259J5-00   It asserts a GRANT for REQ1 if it is
asserted, or for REQ0 if it is asserted and REQ1 is not asserted
assuming no higher priority request is asserted (REQ9 - REQ2) and the
enable inputs (pins 1 and 2) are in the proper state   Pin 17 generates
a REQUEST ACTIVE output which is asserted whenever any of the ten
GRANTs is asserted

This PAL is used on the LO111 (P ioj) module of the NSC70
The base part number for this PAL is 23-000J5-03
```

Figure 2-8 23-259J5-00

```
T    PAL16L8
P    23-259J5-00
N    HARDWARE ENGINEER
D    23-MAR-83
S    01 02 03 04 05 06 07 08 09 GND
     11 12 13 14 15 16 17 18 19 VCC
B    IF [VCC] /19 = /01 * 02 * /03 * 04 * 05 * 06 * 07 * 08 * 09 * 11
     IF [VCC] /18 = /01 * 02 * /05 * 06 * 07 * 08 * 09 * 11
     IF [VCC] /17 = /01 * 02 * /06 * 07 * 08 * 09 * 11
     IF [VCC] /16 = /01 * 02 * /07 * 08 * 09 * 11
     IF [VCC] /15 = /01 * 02 * /08 * 09 * 11
     IF [VCC] /14 = /01 * 02 * /09 * 11
     IF [VCC] /13 = /01 * 02 * /11
     IF [VCC] /12 = /01 * 02 * /04 * 05 * 06 * 07 * 08 * 09 * 11
E    END OF EQUATIONS
TITLE   High Order Bus Arbitrator
This PAL is used for both the CONTROL BUS and the DATA BUS ARBITRATORS.
in conjunction with 23-258J5-00  It asserts a GRANT for the highest
priority asserted REQ (REQ9 - REQ2) if the enable pins (pins 1 and 2)
are in the correct state
This PAL is used on the LOlll (P ioj) module of the HSC70
The base part number for this PAL is 23-000J5-03
```

Figure 2-9 23-260J5-00

```
T    PAL16L8
P    23-260J5 00
N    HARDWARE ENGINEER
D    3-APR-80

S    01 02 03 04 05 06 07 08 09 GND
     11 12 /P18_RD_HI_EAR /P18_RD_LO_EAR /P18_RD_REPLY_0 /P18_RD_WBUS REG
     /P18_RD_WIND_REG /P18_RD_WADR_REG /19 VCC

B    IF [VCC] /19 = 01 * /02 * /03 * 04 * /05 * 09
     .ENABLES DECODER FOR ADDRESSES 17 770 040 - 056

     IF [VCC] /P18_RD_WADR_REG = 01 * /02 * /03 * /04 * /05 * 09
     .ENABLES ADDRESSES  17 770 000 - 016

     IF [VCC] /P18_RD_WIND_REG = 01 * /02 * /03 * /04 * 05 * /06
     * /07 * /08 * 09
     .DECODES 17 770 020

     IF [VCC] /P18_RD_WBUS_REG = 01 * /02 * /03 * /04 * 05 * /06
     * /07 * 08 * 09
     .DECODES 17 770 022

     IF [VCC] /P18_RD_REPLY_0 = 01 * /02 * /03 * 04 * /05 * 09
     + 01 * /02 * /03 * /04 * /05 * 09
     + 01 * /02 * /03 * /04 * 05 * /06 * /07 * /08 * 09
     + 01 * /02 * /03 * /04 * 05 * /06 * /07 * 08 * 09
     + 01 * /02 * /03 * /04 * 05 * /06 * 07 * /08 * 09
     + 01 * /02 * /03 * /04 * 05 * /06 * 07 * 08 * 09
     .INCLUSIVE OR OF PREVIOUS FOUR TERMS AND THE FOLLOWING TWO TERMS

     IF [VCC] /P18_RD_LO_EAR = 01 * /02 * /03 * /04 * 05 * /06 * 07 * /08
     * 09
     .DECODES 17 770 024

     IF [VCC] /P18_RD_HI_EAR = 01 * /02 * /03 * /04 * 05 * /06 * 07 * 08 * 09
     .DECODES 17 770 026

E    END OF EQUATIONS

TITLE   E91  IO PAGE RD DECODER

This PAL is used on the LO111 (P 10J) module of the HSC70

The base part number for this PAL is 23-000J5-04
```


42

Figure 2-10 23-261J5-00

```
T    PAL16L8
P    23-261J5-00
N    TOM FAVA
D    3-APR-80
S    01 02 03 04 05 06 07 08 09 GND
     11 12 /P18_LD_REPLY_0 /P18_LD_KST_REG
     /P18_LD_KIN_REG /P18_LD_SWD_REG /P18_LD_PIO_CSR
     /P18_LD_WIND_REG /P18_LD_WADR_REG VCC
B    IF [VCC] /P18_LD_WADR_REG = 01 • /02 • /03 • /04 • /05 • 09
     .ENABLES ADDRESSES 17 770 000 - 016
     IF[VCC] /P18_LD_WIND_REG = 01 • /02 • /03 • /04 • 05 • /06 • /07
     • /08 • 09
     .ADDRESS 17 770 020
     IF [VCC] /P18_LD_PIO_CSR = 01 • /02 • /03 • 04 • /05 • /06 • /07
     • /08 • 09
     .ADDRESS 17 770 040
     IF [VCC] /P18_LD_SWD_REG = 01 • /02 • /03 • 04 • /05 • /06 • /07
     • 08 • 09
     .ADDRESS 17 770 042
     IF [VCC] /P18_LD_KIN_REG = 01 • /02 • /03 • 04 • /05 • /06 • 07
     • /08 • 09
     .ADDRESS 17 770 044
     IF [VCC] /P18_LD_KST_REG = 01 • /02 • /03 • 04 • /05 • /06 • 07
     • 08 • 09
     .ADDRESS 17 770 046
     IF [VCC] /P18_LD_REPLY_0 = 01 • /02 • /03 • /04 • /05 • 09
     • 01 • /02 • /03 • /04 • 05 • /06 • /07 • /08 • 09
     • 01 • /02 • /03 • 04 • /05 • /06 • 09
E    END OF EQUATIONS
IO PAGE WRITE DECODE
This PAL is used on the LO111 (P 10J) module of the HSC70
The base part number for this PAL is 23-000J5-04
```

Figure 2-11 23-262J5-00

```
T    PAL16L8
P    23-262J5-00
N    HARDWARE ENGINEER
D    23-FEB-85

S    RO R1 R2 R3 (5) (6) (7) (8) (9) GND
     TIAK IN_BUS4 /INTR_EN TIAKO /EN_DL_CLR IN_BUS7 IN_BUS6 IN_BUS5 IN_BUS3
     VCC

B    IF [INTR_EN] /IN_BUS3 = R2
                           + /R1

     IF [INTR_EN] /IN_BUS4 = /R3 * /R2
                           + R2 * R1

     IF [INTR_EN] /IN_BUS5 = R3
                           + /R2
                           + R1

     IF [INTR_EN] /IN_BUS6 = R2 * /R1

     IF [INTR_EN] /IN_BUS7 = R3
                           + R2 * /R1

     IF [VCC] /INTR_EN = /R2 * TIAK
                       + /R3 * TIAK

     IF [VCC] /TIAKO = /RO
                     + /R1
                     + /R2
                     + /R3
                     + /TIAK

     IF [VCC] /EN_DL_CLR = /R3 * INTR_EN
```

(Continued on next page)

Figure 2-11 (Cont) 23-262J5-00

```
E    END OF EQUATIONS

E123, INTERRUPT VECTOR GENERATOR

This PAL is used on the L0111 (P ioj) module of the HSC70

The base part number for this PAL is 23-000J5-04

This PAL generates bits 07 03 of the interrupt vector for interrupts
generated on this module, vector bits 02 00 are generated externally
to the PAL   The interrupts, defined by inputs R3 0, and vectors are
shown below

                          R 3 0   VECTOR
   CONTROL BUS 7          HLHH    134
   CONTROL BUS 6          HLHL    130
   CONTROL BUS 5          HLLH    124
   TERMINAL RCVR          LHLL    060
   TERMINAL XMTR          LHLH    064
   AUX1 RCVR              LHHL    300
   AUX1 XMTR              LLLH    304
   AUX2 RCVR              LLHL    310
   AUX2 XMTR              LLHH    314
   CONTROL BUS  4         HLLL    120

Output INTR_EN is used to enable other parts of the interrupt logic
during an interrupt cycle

Output TIAKO propagates the Interrupt Acknowledge signal to the Program
Memory Bus in the event a Level 4 interrupt is generated external to
the module (ie, by the RX33 controller)

Output EN_DL_CLR is used to enable a clear signal to the DLART
interrupt output currently being serviced
```

Figure 2-12 23-263J5-00

```
T    PAL16L8
P    23-263J5-00
N    HARDWARE ENGINEER
D    30-OCT-85
S    R3 R2 R1 R0 (5) RIRQ4_IR /TINIT /IR_EN (9) GND
     (11) IRQ_5 /CLR_C_IR4 /CLR_C_IR5 /CLR_C_IR6 /CLR_C_IR7 IRQ 7 IRQ_6
     IRQ_4 VCC
B    IF [VCC] /IRQ 4 = /R3 * /R2 * /R1 * /R0
                     + /R3 *  R2 *  R1 *  R0
                     +  R3 *  R2 * /R0
                     +  R3 * /R2 *  R1
                     + /RIRQ4_IR *  R3 *  R1
                     +  R3 * /R1 *  R0

     IF [VCC] /IRQ_5 = /R3
                     +  R2
                     +  R1
                     + /R0

     IF [VCC] /IRQ_6 = /R3
                     +  R2
                     + /R1
                     +  R0

     IF [VCC] /IRQ_7 = /R3
                     +  R2
                     + /R1
                     + /R0

     IF [VCC] /CLR_C_IR4 * R3 * /R2 * /R1 * /R0 * IR_EN * TINIT
     IF [VCC] /CLR_C_IR5 = R3 * /R2 * /R1 *  R0 * IR_EN * TINIT
     IF [VCC] /CLR_C_IR6 = R3 * /R2 *  R1 * /R0 * IR_EN * TINIT
     IF [VCC] /CLR_C_IR7 = R3 * /R2 *  R1 *  R0 * IR_EN * TINIT
E    END OF EQUATIONS
```

E126. INTERRUPT DECODER

This PAL is used on the LO111 (P 10J) module of the HSC70

The base part number for this PAL is 23-000J5 04

This PAL generates the four INTERRUPT LEVEL 7 4 inputs to the J-11 chip based on the R3 0 inputs and the Program Memory Bus IRQ4 input It also generates a clear signal to the flipflop holding the Control Bus Interrupt currently being serviced

4(a

Figure 2-13 23-264J5-00

```
T    PAL16L8
P    23 264J5-00
N    HARDWARE ENGINEER
D    02-JAN-85

S    1 2 3 4 5 6 7 8 9 GND
     11 /12 /TEST /14 /15 /16 /17 /18 /19 VCC

B    IF [VCC] 12 = 1 * 2 * 3 * 4 * 5 * /6 * 7 * 11
     + 1 * 2 * 3 * 4 * 5 * 6 * 7 * /8 * 11

     IF [VCC] 14 = 1 * 2 * 3 * 4 * 5 * /6 * 7 * 8

     IF [VCC] 15 = 1 * 2 * 3 * 4 * 5 * /6 * 7 * /8

     IF [VCC] 16 = 1 * 2 * 3 * 4 * 5 * 6 * 7 * /8

     IF [VCC] 17 = 1 * /2 * 3 * 4 * /TEST
                 + 1 * 2 * /4 * /TEST
                 + 1 * 2 * /3 * 4 * /TEST

     IF [VCC] 18 = 1 * 2 * 3 * 4 * 5 * /6 * 7 * 9
                 + 1 * 2 * 3 * 4 * 5 * 6 * 7 * /8 * 9

     IF [VCC] 19 = 1 * /2 * 3 * 4 * 9 * /TEST
                 + 1 * 2 * /4 * 9 * /TEST
                 + 1 * 2 * /3 * 4 * 9 * /TEST
                 + 1 * 2 * 3 * 4 * 5 * /6 * 7
                 + 1 * 2 * 3 * 4 * 5 * 6 * 7 * /8

E    END OF EQUATION
```

E186 Bibus Address Decoder

This PAL is used on the LO111 (P ioj) module of the HSC70

The base part number for this PAL is 23-000J5-04

This PAL generates RD and WRT enables for these elements on the Bibus as well as for the Bibus transceivers These elements include the Bootstrap PROM and the DLARTs for the Console Terminal and the AUX1 and AUX2 serial ports A TEST input is provided to disable the bootstrap PROMs and allow the module tester to supply a specialized bootstrap program for testing the module

Figure 2-14 23-007K7-00

```
T    PAL20X8
P    23 007K7-00
N    HARDWARE ENGINEER
D    17-DEC-86

S    CLK  IO D0 D1 D2 D3 D4 D5 D6 D7  I1 GND
     /OC /MAX Q7 Q6 Q5 Q4 Q3 Q2 Q1 Q0 /CI VCC

B    /Q0  =  /I1*/IO                          .CLEAR LSB
             +      IO*/Q0                    .COUNT/HOLD
             +   I1*/IO*/D0                   .LOAD D0 (LSB)
             +   I1* IO* CI                   .COUNT

     /Q1  =  /I1*/IO                          .CLEAR
             +      IO*/Q1                    .COUNT/HOLD
             +   I1*/IO*/D1                   .LOAD D1
             +   I1* IO* CI*Q0                 COUNT

     /Q2  =  /I1*/IO                          .CLEAR
             +      IO*/Q2                    .COUNT/HOLD
             +   I1*/IO*/D2                   .LOAD D2
             +   I1* IO* CI*Q0*Q1              .COUNT

     /Q3  =  /I1*/IO                          .CLEAR
             +      IO*/Q3                    .COUNT/HOLD
             +   I1*/IO*/D3                   .LOAD D3
             +   I1* IO* CI*Q0*Q1*Q2           COUNT

     /Q4  =  /I1*/IO                          CLEAR
             +      IO*/Q4                    .COUNT/HOLD
             +   I1*/IO*/D4                   LOAD D4
             +   I1* IO* CI*Q0*Q1*Q2*Q3       COUNT

     /Q5  =  /I1*/IO                          .CLEAR
             +      IO*/Q5                    COUNT/HOLD
             +   I1*/IO*/D5                   .LOAD D5
             +   I1* IO* CI*Q0*Q1*Q2*Q3*Q4    .COUNT

     /Q6  =  /I1*/IO                          .CLEAR
             +      IO*/Q6                    COUNT/HOLD
             +   I1*/IO*/D6                   LOAD D6
             +   I1* IO* CI*Q0*Q1*Q2*Q3*Q4*Q5 .COUNT

     /Q7  =  /I1*/IO                          .CLEAR MSB
             +      IO*/Q7                    .COUNT/HOLD
             +   I1*/IO*/D7                   .LOAD D7 (MSB)
             +   I1* IO* CI*Q0*Q1*Q2*Q3*Q4*Q5*Q6  COUNT

IF [VCC]  MAX = Q0*Q1*Q2*Q3*Q4*Q5*Q6*Q7       .MAX COUNT
```

(Continued on next page)

Figure 2-14 (Cont) 23-007K7-00

```
E   E137  MABO (Memory Address Register 0) PAL

THIS IS AN 8 BIT SYNCHRONOUS COUNTER WITH PARALLEL LOAD  CLEAR  AND HOLD
CAPABILITY   THE LOAD OPERATION LOADS THE INPUTS (D7-D0) INTO THE OUTPUT
REGISTER (Q7-Q0)   THE CLEAR OPERATION RESETS THE OUTPUT REGISTER TO ALL
LOWS  THE HOLD OPERATION HOLDS THE PREVIOUS VALUE REGARDLESS OF CLOCK
TRANSITIONS  THE INCREMENT OPERATION ADDS ONE TO THE OUTPUT REGISTER
WHEN THE CARRY IN IS TRUE (/CI-L)  OTHERWISE THE OPERATION IS A HOLD
MAX IS TRUE (/MAX-L) WHEN THE OUTPUT REGISTER (Q7 Q0) IS
ALL HIGHS

OPERATIONS TABLE

    CLK  I1  I0  /CI  D7 D0  Q7 Q0      OPERATION
     X   X   X    X     X       Z       HI-Z
     C   L   L    X     X       L       CLEAR
     C   L   H    X     X       Q       HOLD
     C   H   L    X     D       D       LOAD
     C   H   H    H     X       Q       HOLD
     C   H   H    L     X   Q PLUS 1    INCREMENT

This PAL is used on the LO117 (M std2) module of the MSC70

The base part number for this PAL is 23 000K7-01
```

Figure 2-15 23-008K7 00

```
T    PAL20X8
P    23 008K7 00
N    HARDWARE ENGINEER
D    17-DEC 85

S    CLK  IO D0 D1 D2 D3 D4 D5 D6 D7  I1 GND
     /OC /MAX Q7 Q6 Q5 Q4 Q3 Q2 Q1 Q0 CI VCC

B    /Q0  =  /I1*/IO                          .CLEAR LSB
             +      IO*/Q0                    .COUNT/HOLD
             +   I1*/IO*/D0                   .LOAD D0 (LSB)
             +   I1* IO* CI                   .COUNT

     /Q1  =  /I1*/IO                          .CLEAR
             +      IO*/Q1                    .COUNT/HOLD
             +   I1*/IO*/D1                   .LOAD D1
             +   I1* IO* CI*Q0                 COUNT

     /Q2  =  /I1*/IO                          .CLEAR
             +      IO*/Q2                    .COUNT/HOLD
             +   I1*/IO*/D2                   .LOAD D2
             +   I1* IO* CI*Q0*Q1              .COUNT

     /Q3  =  /I1*/IO                          .CLEAR
             +      IO*/Q3                    .COUNT/HOLD
             +   I1*/IO*/D3                   .LOAD D3
             +   I1* IO* CI*Q0*Q1*Q2           .COUNT

     /Q4  =  /I1*/IO                          .CLEAR
             +      IO*/Q4                    .COUNT/HOLD
             +   I1*/IO*/D4                   .LOAD D4
             +   I1* IO* CI*Q0*Q1*Q2*Q3        .COUNT

     /Q5  =  /I1*/IO                          .CLEAR
             +      IO*/Q5                    .COUNT/HOLD
             +   I1*/IO*/D5                   .LOAD D5
             +   I1* IO* CI*Q0*Q1*Q2*Q3*Q4     .COUNT

     /Q6  =  /I1*/IO                          .CLEAR
             +      IO*/Q6                    .COUNT/HOLD
             +   I1*/IO*/D6                   .LOAD D6
             +   I1* IO* CI*Q0*Q1*Q2*Q3*Q4*Q5  .COUNT

     /Q7  =  /I1*/IO                          .CLEAR MSB
             +      IO*/Q7                    .COUNT/HOLD
             +   I1*/IO*/D7                   .LOAD D7 (MSB)
             +   I1* IO* CI*Q0*Q1*Q2*Q3*Q4*Q5*Q6  .COUNT

IF [VCC]  MAX = Q0*Q1*Q2*Q3*Q4*Q5*Q6*Q7       .MAX COUNT
```

(Continued on next page)

Figure 2-15 (Cont.) 23-008K7-00

E E143 MAR1 (Memory Address Register 1) PAL

This is an 8-bit synchronous counter with parallel load, clear, and hold capability The load operation loads the inputs (D7-D0) into the output register (Q7-Q0) The clear operation resets the output register to all lows the hold operation holds the previous value regardless of clock transitions the increment operation adds one to the output register when the carry-in is true (CI=H), otherwise the operation is a hold Max is true (/MAX=L) when the output register (Q7-Q0) is all highs

Operations Table

CLK	I1	I0	CI	D7-D0	Q7-Q0	OPERATION
X	X	X	X	X	Z	HI-Z
C	L	L	X	X	L	CLEAR
C	L	H	X	X	Q	HOLD
C	H	L	X	D	D	LOAD
C	H	H	L	X	Q	HOLD
C	H	H	H	X	Q PLUS 1	INCREMENT

This PAL is used on the LO117 (M std2) module of the HSC70

The base part number for this PAL is 23-000K7-01

Figure 2-16 23-009K7-00

```
T    PAL20X8
P    23-009K7-00
N    HARDWARE ENGINEER
D    18-DEC-86

S    CLK  IO DO D1 D2 D3 D4 D5 D6 D7  I1 GND
     /OC CIN Q7 Q6 Q5 Q4 Q3 Q2 Q1 Q0 /ENB VCC

B    /Q0  =  /I1*/IO                          .CLEAR LSB
             +      IO*/Q0                    .COUNT/HOLD
             +   I1*/IO*/D0                   .LOAD D0 (LSB)
             +   I1* IO* ENB*CIN                   .COUNT

     /Q1  =  /I1*/IO                          .CLEAR
             +      IO*/Q1                    .COUNT/HOLD
             +   I1*/IO*/D1                    LOAD D1
             +   I1* IO* ENB*CIN*Q0                .COUNT

     /Q2  =  /I1*/IO                          .CLEAR
             +      IO*/Q2                    .COUNT/HOLD
             +   I1*/IO*/D2                   .LOAD D2
             +   I1* IO* ENB*CIN*Q0*Q1             .COUNT

     /Q3  =  /I1*/IO                          .CLEAR
             +      IO*/Q3                    .COUNT/HOLD
             +   I1*/IO*/D3                   .LOAD D3
             +   I1* IO* ENB*CIN*Q0*Q1*Q2          .COUNT

     /Q4  =  /I1*/IO                          .CLEAR
             +      IO*/Q4                    .COUNT/HOLD
             +   I1*/IO*/D4                   .LOAD D4
             +   I1* IO* ENB*CIN*Q0*Q1*Q2*Q3       .COUNT

     /Q5  =  /I1*/IO                          .CLEAR
             +      IO*/Q5                     COUNT/HOLD
             +   I1*/IO*/D5                   .LOAD D5
             +   I1* IO* ENB*CIN*Q0*Q1*Q2*Q3*Q4    .COUNT

     /Q6  =  /I1*/IO                          .CLEAR
             +      IO*/Q6                    .COUNT/HOLD
             +   I1*/IO*/D6                   .LOAD D6

     /Q7  =  /I1*/IO                          .CLEAR MSB
             +      IO*/Q7                    .COUNT/HOLD
             +   I1*/IO*/D7                   .LOAD D7 (MSB)
```

(Continued on next page)

Figure 2-16 (Cont) 23-009K7-00

E E150 MAR2 (Memory Address Register 2) PAL

This is a 6-bit synchronous counter with parallel load, clear, and hold capability The load operation loads the inputs (D7-D0) into the output register (Q7-Q0) The clear operation resets the output register to all lows The hold operation holds the previous value regardless of clock transitions The increment operation adds one to the output register when the enable is true (/ENB=L) and carry-in is true (CIN=H) Otherwise the operation is a hold Bits 6 & 7 are used only as register bits and do not increment

Operations Table

/OC	CLK	I1	I0	/ENB	CIN	D7-D0	Q5-Q0	Q7-Q6	OPERATION
H	X	X	X	X	X	X	Z	Z	HI-Z
L	C	L	L	X	X	X	L	L	CLEAR
L	C	L	H	X	X	X	Q	Q	HOLD
L	C	H	L	X	X	D	D	D	LOAD
L	C	H	H	H	X	X	Q	Q	HOLD
L	C	H	H	X	L	X	Q	Q	HOLD
L	C	H	H	L	H	X	Q PLUS 1	Q	INCREMENT/HOLD

This PAL is used on the LO117 (M std2) module of the MSC70

The base part number for this PAL is 23-000K7-01

Figure 2-17 23-127k5-00

```
T    PAL16R8
P    23-127K5-00
D    28-AUG-85
S    PAL_CLK /BINIT /REG_SEL_0 /REG_SEL_2 /REG_SEL_4 /REG_SEL_6
      /INWD SYNC OUT_CYC PIN9 GND
      /TS_OE /REG_RD /RD_MAR_2 /RD_MAR_1 /RD_MAR_0
      /MAR_LOAD_DONE /LD_MAR_2 /LD_MAR_1 /LD_MAR_0 VCC
B    LD_MAR_0 = REG SEL_2 • SYNC_OUT_CYC • /MAR_LOAD_DONE • /BINIT
      • BINIT • /MAR LOAD_DONE

      LD_MAR_1 = REG SEL_4 • SYNC_OUT_CYC • /MAR_LOAD_DONE • /BINIT
      • BINIT • /MAR LOAD_DONE

      LD_MAR_2 = REG_SEL_6 • SYNC_OUT_CYC • /MAR_LOAD_DONE • /BINIT
      • BINIT • /MAR_LOAD_DONE

      MAR_LOAD_DONE = REG_SEL_2 • SYNC_OUT_CYC • /BINIT
      • REG_SEL_4 • SYNC_OUT_CYC • /BINIT
      • REG SEL_6 • SYNC_OUT_CYC • /BINIT
      • MAR_LOAD_DONE • SYNC_OUT_CYC • /BINIT
      • BINIT

      RD_MAR_0 = REG_SEL_2 • INWD • /BINIT

      RD_MAR_1 = REG_SEL_4 • INWD • /BINIT

      RD_MAR_2 = REG_SEL_6 • INWD • /BINIT

      REG_RD =  REG_SEL_0 • INWD
      • REG_SEL_2 • INWD
      • REG_SEL_4 • INWD
      • REG_SEL_6 • INWD
```

E E120 REGCTL (Register Control) PAL for RX33 Controller

This PAL generates signals which control the reading and writing of the floppy registers primarily the MAR, that are external to the FDC chip All outputs are asserted low

BINIT will cause all LD MAR x outputs to become asserted in order to clear the MAR, (especially the Module OK bit), on power-up

Figure 2-18 23-128K5-00

```
T    PAL16R8
P    23-128K5-00
D    28-AUG-85
S    PAL_CLK /RESET /REG_SEL_0 /REG_SEL_2 /REG_SEL_4 /REG_SEL_6
      /INWD SYNC_OUT_CYC /DISK_RW GND
      /TS_OE /RX_LOAD_DONE RX_SEL_0 /RD_CSR /LD_CSR /RX_SLAVE_WR
      /RX_SLAVE_RD RX_SEL_1 PIN19 VCC
B    /RX_SEL_0 = /REG_SEL_2 * /REG_SEL_6 * /DISK_RW
     /RX_SEL_1 = /REG_SEL_4 * /REG_SEL_6 * /DISK_RW
     RX_SLAVE_RD =    REG_SEL_0 * INWD * /RESET
      +  REG SEL 2 * INWD * /RESET
      +  REG SEL 4 * INWD * /RESET
      +  REG_SEL_6 * INWD * /RESET
     RX_SLAVE_WR =    REG_SEL_0 * SYNC_OUT_CYC * /RX_LOAD_DONE * /RESET
      +  REG SEL_2 * SYNC_OUT_CYC * /RX_LOAD_DONE * /RESET
      +  REG_SEL_4 * SYNC_OUT_CYC * /RX_LOAD_DONE * /RESET
      +  REG_SEL_6 * SYNC_OUT_CYC * /RX_LOAD_DONE * /RESET
     RX_LOAD_DONE = REG_SEL_0 * SYNC_OUT_CYC
      + REG_SEL_2 * SYNC_OUT_CYC
      + REG_SEL_4 * SYNC_OUT_CYC
      + REG_SEL_6 * SYNC_OUT_CYC
      + RX_LOAD_DONE * SYNC_OUT_CYC
     LD_CSR = REG_SEL_0 * SYNC_OUT_CYC * /RX_LOAD_DONE
     RD_CSR = REG_SEL_0 * INWD * /RESET
C    E121. RXCTL (RX Control) PAL for RX33 Controller
This PAL generates signals which control the reading and writing
of the floppy registers inside the FDC chip  All outputs are
asserted low except RX_SEL_0,1 which are asserted high
RX_SEL_0 and RX_SEL_1 are both asserted whenever DISK_RW is
asserted since only the RX DBUFF is used during the actual
read/write transfers to the floppy drive
This PAL also controls the reading and writing of the CSR hi byte
which is external to the FDC chip
```

Figure 2-19 23-131K5-00

```
T    PAL16R8
P    23-131K5-00
D    28-AUG-85

S    PAL_CLK BYTE_REQ PIN3 TDIN /R_RPLY /DISK WRITE MASTER /DMA_TEST PIN9
     GND /TS_OE /ENB_RBUF /RBUF_EMPTY /DMA_RD_REQ /ENB_RBUF_LB /ENB_RBUF HB
     /LD_RBUF /RX_MASTER_WR /STOP_DMA_WR TEST VCC

B    RX_MASTER WR = DISK_WRITE * LD_RBUF * /RX_MASTER_WR * RBUF_EMPTY
      + DISK_WRITE * BYTE_REQ * /RX_MASTER_WR * /RBUF_EMPTY
      * /DMA_TEST

     LD_RBUF = DISK_WRITE * MASTER * TDIN * R_RPLY * RBUF EMPTY * /LD RBUF

     ENB_RBUF LB = DISK WRITE * LD_RBUF * /ENB_RBUF LB
      + DISK_WRITE * ENB_RBUF_LB * RX_MASTER_WR

     ENB RBUF_HB = DISK_WRITE * BYTE_REQ * /RBUF_EMPTY * /ENB_RBUF_HB
      * /ENB_RBUF_LB * /DMA TEST
      + DISK_WRITE * ENB_RBUF HB * RX MASTER_WR

     DMA_RD_REQ = DISK_WRITE * BYTE_REQ * RBUF_EMPTY * /DMA_RD_REQ
      * /MASTER * /ENB_RBUF_HB * /DMA TEST
      + DISK_WRITE * DMA_RD REQ * /MASTER
      + DISK_WRITE * /STOP_DMA WR TEST * RBUF_EMPTY * /DMA_RD_REQ
      * /MASTER * /ENB RBUF_HB * DMA_TEST

     RBUF_EMPTY = /DISK_WRITE
      + DISK WRITE * /LD RBUF * RBUF_EMPTY
      + DISK_WRITE * /RBUF_EMPTY * /RX_MASTER WR * ENB_RBUF_HB
      * /DMA_TEST
      + DISK_WRITE * /RBUF_EMPTY * DMA TEST * ENB_RBUF LB
      * /RX_MASTER_WR

     ENB_RBUF = DISK WRITE * LD_RBUF * /ENB_RBUF
      + DISK_WRITE * BYTE_REQ * /RBUF_EMPTY * /ENB_RBUF
      * /ENB_RBUF_LB * /DMA_TEST
      + DISK_WRITE * ENB_RBUF * RX_MASTER_WR

     STOP_DMA_WR_TEST = DISK_WRITE * /DMA_TEST
      + STOP_DMA_WR_TEST * DISK_WRITE
      + DMA_TEST * MASTER

E    E136, DISKWR (Disk Write) Sequencer PAL for RX33 Controller

This PAL generates signals steer the data
path during writes to the floppy disk   DMA test mode allows data
to be transferred from memory to the sector register  Only the
lower byte is used for DMA test   All outputs are asserted low
```


56

Figure 2-20 23-143K5-00

```
T    PAL16R8
P    23-143K5 00
N    HARDWARE ENGINEER
D    10-JAN-86

S    PAL_CLK RX_D07 RX_D06 RX D05 RX_D04 INTR REQ /LD_CSR /RESET RX_D12
     GND /TS_OE /INTR DLY /INTR DONE /DISK_WRITE /DISK_READ /LD_CSR_DLY
     /DISK_RW /FORMAT PIN19 VCC

B    DISK_READ = LD_CSR * /RX_D12 * RX_D07 * /RX_D06 * /RX_D05 * /RESET
     + LD_CSR * /RX_D12 * RX_D07 * RX_D06 * /RX_D05 * /RX_D04 * /RESET
     + LD_CSR * /RX_D12 * RX_D07 * RX_D06 *  RX_D05 * /RX_D04 * /RESET
     + DISK_READ * LD_CSR_DLY

     DISK_WRITE = LD_CSR * /RX_D12 * RX_D07 * /RX_D06 * RX_D05 * /RESET
     + LD_CSR * /RX_D12 * RX D07 * RX_D06 * RX_D05 * RX_D04 * /RESET
     + DISK_WRITE * LD_CSR_DLY
     + FORMAT * /INTR_DONE

     DISK_RW = DISK_READ
          + DISK_WRITE * /INTR_REQ * /INTR_DLY * /INTR_DONE

     LD_CSR_DLY = LD_CSR * /LD_CSR_DLY * /RESET
          + LD_CSR_DLY * /INTR_REQ * /RESET

     FORMAT = LD_CSR * /RX_D12 * RX_D07 * RX_D06 * RX_D05 * RX_D04 * /RESET
          + FORMAT * /INTR_DONE * /RESET

     INTR_DLY = INTR_REQ

     INTR_DONE = INTR_DLY * /INTR_REQ

C    C129, FUNCDEC2 (Function Decode) PAL for RX33 Controller

PAL generates signals which establish the mode of operation of
the disk read and disk write PALs  A read or write mode remains
in effect from the time the command is loaded into the CSR
until an interrupt or reset occurs  If a disk read or write is in
effect then DISK_RW is used by the RX_CTL PAL to force the data buffer
to be selected  All outputs are asserted low

FUNDEC2 adds output signals FORMAT, INTR_DLY, and INTR_DONE and changes
equations for DISK_WRITE and DISK_RW slightly  New equations solve
problems which sometimes occur during formatting commands which
could cause floppy controller to write bad parity to a single memory
location if index pulse occurred while floppy was in the middle of a
memory read cycle  New equations now cause controller to wait for
trailing edge of interrupt instead of using leading edge to terminate
the format operation

This PAL is used on the LO117 (N std2) module of the HSC70

The base part number for this PAL is 23-000K5-04
```

Figure 2-21 23-144K5-00

```
T    PAL16R8
P    23-144K5 00
N    HARDWARE ENGINEER
D    7-JAN-86

S    PAL CLK BYTE REQ PIN3 MASTER /DISK_READ PIN6 PIN7 /DMA_TEST PIN9
     GND /TS OE /DMA_TEST_FLAG /STOP DMA TEST /LB_LOADED /DMA_WR_REQ
     /RX_MASTER_RD /LD_XBUF_HB /LD_XBUF_LB PIN19 VCC

B    RX_MASTER RD = DISK_READ * BYTE_REQ * /LD_XBUF_LB * /LD XBUF_HB
          * /DMA_TEST * /DMA TEST FLAG * /DMA_WR_REQ * /MASTER
          + DISK_READ * DMA_TEST * /LD_XBUF_LB * /LD XBUF HB
          * /DMA_WR_REQ * /MASTER * /STOP_DMA TEST
          + DISK_READ * RX_MASTER_RD * /LD_XBUF_LB * /LD_XBUF_HB

     LD_XBUF_LB = DISK_READ * RX_MASTER RD * /LB_LOADED * /LD_XBUF_LB

     LD_XBUF_HB = DISK_READ * RX MASTER_RD *  LB_LOADED * /LD_XBUF HB

     DMA_WR_REQ = DISK_READ * RX_MASTER_RD * LD_XBUF_HB * /MASTER
          * /DMA_WR_REQ
          + DISK_READ * DMA_WR_REQ * /MASTER

     LB_LOADED = DISK_READ * RX_MASTER_RD * LD_XBUF_LB * /LB_LOADED
          + DISK_READ * LB_LOADED * /LD_XBUF_HB

     DMA_TEST_FLAG = DMA_TEST * DISK_READ * /DMA_TEST_FLAG
          + DMA_TEST_FLAG * DISK_READ

     STOP_DMA_TEST = DMA_TEST_FLAG * DISK_READ * /DMA_TEST
          + STOP_DMA_TEST * DISK_READ
          + DMA_TEST_FLAG * MASTER

C    E133. DISKRD2 (Disk Read) Sequencer PAL for RX33 Controller

This PAL generates signals which steer
the data path during reads from the floppy disk  All outputs
are asserted low

DMA_TEST input may be used to generate DMA writes to
memory without a floppy drive cabled up  This aids in debug,
board test, and diagnostic fault isolation

DISKRD2 pattern changes equation for RX_MASTER_RD so that low
byte does not get corrupted when reading floppy while system is
heavily loaded and DMA latency is very slow  Floppy data register
now cannot be read into XBUF until prior bus cycle has completed

This PAL is used on the LO117 (M std2) module of the MSC70

The base part number for this PAL is 23-000K5-04
```

Figure 2-22 23-2495-00

```
T    PAL16L8
P    23-249J5-00
D    28-AUG-85
S    /DISK_READ /DISK_WRITE /REG_RD PIN4 PIN5 PIN6 /DAT_EN ADR_EN
     PIN9 GND PIN11 PIN12 PIN13 PIN14 /ENB_XBUF /XMT PIN17 PIN18
     PIN19 VCC
B    IF [VCC] ENB_XBUF = DISK_READ * DAT_EN

     IF [VCC] XMT = DISK_READ * ADR_EN
      + DISK_WRITE * ADR_EN
      + DISK_READ * DAT_EN
      + REG_RD
E    E135, MISC (Miscellaneous) PAL for RX33 Controller

This PAL generates signals which control the transmit/receive
logic, enable the transmit buffer during disk reads, and
generate DMA requests for both disk reads and writes  All outputs
are asserted low
```

Figure 2-23 23-069K5-00

```
T    PAL16R8
P    23-069K5-00
D    20 SEP 85

S    CLK DISK /ENA /CLR UPPER S1 I1 SO IO GND
      /ENAB IORS FOLLOW 0 IORS FOLLOW 1 VALID_0 VALID 1
      I_DET FOL_0 I_DET_FOL_1 IORS_DET_SFOL 0 IORS_DET_SFOL_1 VCC

B    /IORS FOLLOW 0 = CLR * /IO * /SO

     /IORS FOLLOW 1 = CLR * /I1 * /S1

     /VALID 0 = /DISK * IORS DET SFOL 0 * /SO * IORS_FOLLOW 0
      * IORS_DET SFOL 0 * IO * /IORS_FOLLOW_0 * CLR * /VALID_0 * /IO

     /VALID 1 = /DISK * IORS DET SFOL_1 * /S1 * IORS_FOLLOW 1
      * IORS DET SFOL_1 * I1 * /IORS_FOLLOW_1 * CLR * /VALID_1 * /I1

     /I DET FOL 0 = /IO * DISK * /I DET_FOL 0 * UPPER * ENA
      * IORS_DET_SFOL_0 * DISK * /DISK * /IO * CLR * IORS_FOLLOW_0
      * DISK * /I DET FOL_0
      * DISK * I DET_FOL 0 * SO * /IORS_DET_SFOL 0

     /I DET FOL_1 = /I1 * DISK * /I DET_FOL_1 * UPPER * ENA
      * IORS DET_SFOL_1 * DISK * /DISK * /I1 * CLR * IORS FOLLOW_1
      * DISK * /I_DET_FOL_1
      * DISK * I_DET_FOL_1 * S1 * /IORS_DET_SFOL 1

     /IORS DET_SFOL 0 = IO * DISK * /IORS_DET_SFOL_0 * /IORS DET_SFOL_0
      * DISK * SO * UPPER * IORS DET_SFOL 0 * ENA * DISK * /IO
      * /VALID 0 * DISK * CLR * /IORS_FOLLOW 0 * DISK * /IORS_DET_SFOL_0
      * /DISK * /SO

     /IORS DET SFOL_1 = I1 * DISK * /IORS DET_SFOL_1 * /IORS_DET_SFOL_1
      * DISK * S1 * UPPER * IORS_DET_SFOL_1 * ENA * DISK * /I1
      * /VALID_1 * DISK * CLR * /IORS_FOLLOW 1 * DISK * /IORS_DET_SFOL 1
      * /DISK * /S1

E    These equations latch the occurrence of sector and index pulses
   so that the K sdi microcode can deal with them via polling
```

(Continued on next page)

Figure 2-23 (Cont) 23-069K5-00

This PAL produces four output variables for each of two drives These variables are

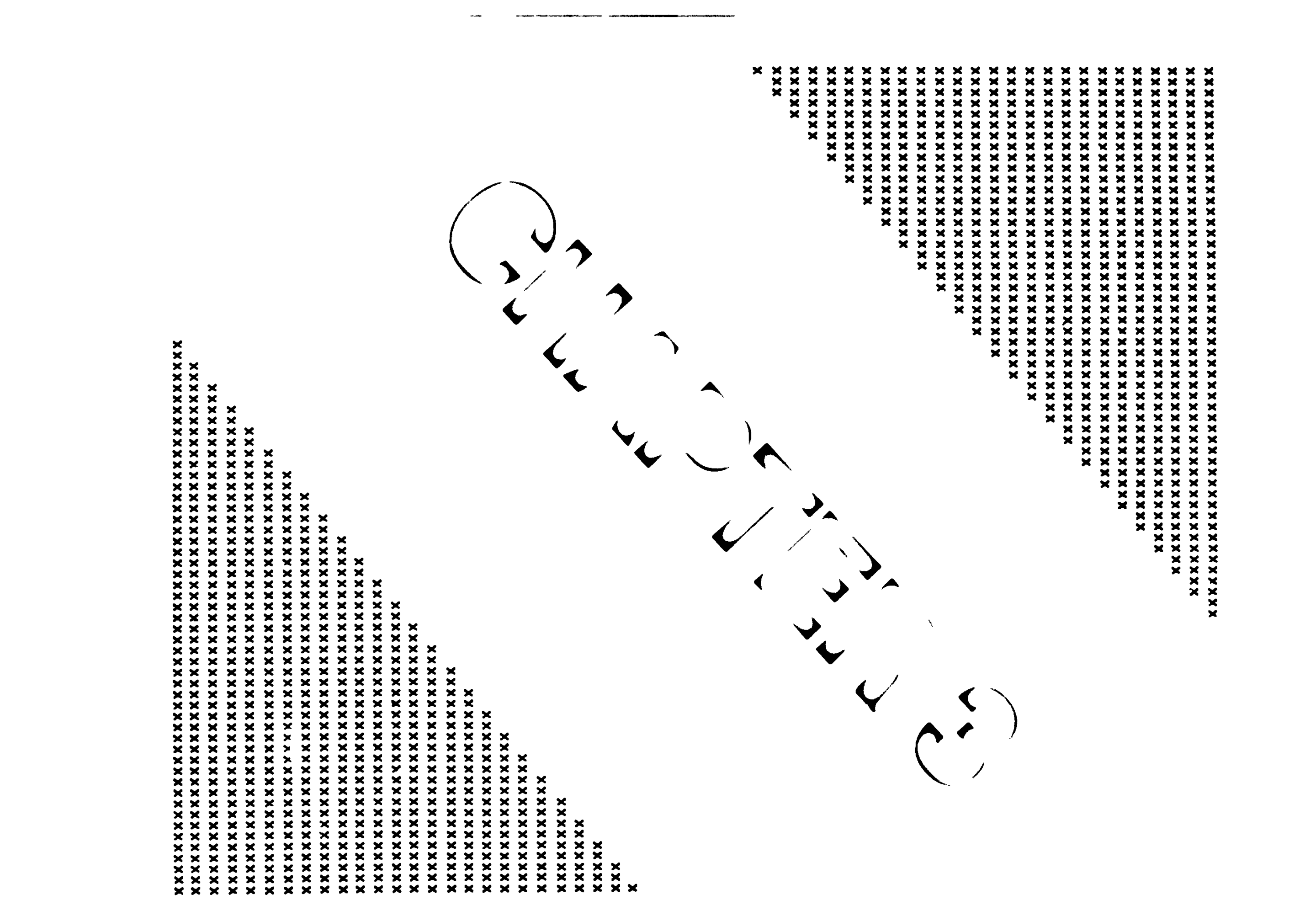
IORS_FOLLOW This is the OR of the drive's Sector Pulse and Index Pulse and is used to do edge-detection on the pulses

IORS_DET SPOL - This is the "I saw something" signal from the PAL to the K sdi microcode None of the other outputs of the PAL are examined unless this bit is set It is set on the falling edge of either Sector or Index Pulse, and cleared whenever it is read by K sdi (while set) or when the PAL is cleared or when the VALID bit goes away

VALID - This indicates if the PAL outputs are valid - i e , if VALID is not asserted then the PAL saw a second pulse before the K sdi microcode read the results of the previous pulse It is set on the rising edge of Index Pulse and cleared whenever the PAL is cleared or whenever a falling edge of Sector Pulse or rising edge of Index Pulse is detected with IORS_DET_SPOL already set

I_DET_FOL - this bit latches the fact that we have seen the rising edge of Index Pulse and the K sdi has not yet been informed It is set on the rising edge of Index Pulse, and cleared if the PAL is cleared or if the PAL is read while IORS_DET_SPOL is set It is also cleared if Sector Pulse is seen and IORS_DET_SPOL has not been set - this overcomes a bug caused by the periodic disabling of drive state receivers by periodic diagnostics in the K sdi microcode

When this PAL is being used by K sti instead of K sdi, the I DET_FOL bit is simply a latched copy of Index Pulse and the IORS DET_SFOL bit is simply a latched copy of sector pulse



CHAPTER 3
HSC50 BOOT PROM LISTINGS

This chapter contains the Boot PROM listings for the HSC50 The page numbers for this chapter are in the upper right hand corner of the listings

TABLE OF CONTENTS

2- 27 ***** HSC50 P.ioc ROM Boot version V01-00 *****
4- 142 Useful Information
5- 199 Useful Information (continued)
6- 231 Program Overview
7- 289 Definitions
7- 290 Address Definitions
8- 336 Bit Definitions
11- 461 Alternate Entry Points and Pointer Table
13- 549 ..TEST 0 - F-11 ISP Test
23- 1097 MEMORY TESTS
24- 1140 ..TEST 1 - Program Memory (Sweep Bits)
25- 1174 ..TEST 2 - Program Memory (Vector area)
26- 1224 ..TEST 3 - Program Memory (Partition area)
27- 1279 FAULT Lamp Routine
28- 1335 ..TEST 4 - TU58 Tests
32- 1559 ..TEST 5 - Transfer Control to Loaded Image
35- 1698 Issue TU58 Sync and Self-Test
36- 1805 Read TU58 Boot Blocks to Program Memory
38- 1869 TU58 Read Routine
39- 1980 Synchronize with a TU58
41- 2059 Send Character to TU58
42- 2117 Receive a Character from a TU58
43- 2187 Send a TU58 Command Packet
46- 2330 Receive a Command or Data Packet from the TU58
53- 2652 Routine to save TU58 error reasons
54- 2686 Moving Inversions Test Description
55- 2726 Moving Inversions Test
56- 2833 Permanent Storage

```
                        ;+
                        ;******************************************************************************
                        ;program name:  MSCS0 ROM Bootstrap
                        ;
                        ;
                              .ident  01,00,< ROM Boot>
                              .title  MSCS0 P.ioc  ROM Boot  V01-00
                              .sbttl  ***** MSCS0 P.ioc  ROM Boot version V01-00 *****
                              .ident  /A0100/
                        ;
                        ;program abstract:
                        ;     This program is designed to execute from the bootstrap ROM located
                        ;     on the P.ioc module.  The bootstrap consists of tests of the F-11
                        ;     CPU, part of the Program P.ioc memory, and the TU58; followed by
                        ;     loading of the next part of the bootstrap sequence from the TU58.
                        ;     The purpose of the bootstrap tests is to provide confidence that
                        ;     the P.ioc can be used to load further software from the TU58.
                        ;     The software loaded as the last action of the bootstrap program
                        ;     consists of more extensive tests of the P.ioc module.  After
                        ;     these tests finish execution, the MSCS0 Functional Software is
                        ;     loaded and initiated.
                        ;
                        ;     Two version of the boot ROM can be created from this source:
                        ;     1) The P.ioc version
                        ;     2) The P.xxx (Secondary P) version (define SECNDP)
                        ;
                        ;copyright statement:
                        ;     Copyright 1982, Digital Equipment Corporation
                        ;
                        ;disclaimer:
                        ;     The information in this document is subject to change without
                        ;     notice and should not be construed as a commitment by Digital
                        ;     Equipment Corporation.  Digital Equipment Corporation assumes
                        ;     no responsibility for any errors that may appear in this doc-
                        ;     ument.
                        ;
                        ;     The software described in this document is furnished under a
                        ;     license and may only be used or copied in accordance with the
                        ;     terms of such license.
                        ;
                        ;     Digital Equipment Corporation assumes no responsibility for the
                        ;     use of reliability of its software on equipment that is not sup-
                        ;     plied by Digital.
                        ;
                        ;author - version - date list
                        ;
                        ;     Mike Mare      V01-00  SEP 1982
                        ;
                        
                              .mcall  hminit,kg6
 69 000000                    hminit
 70 000000                    kg6
```

```
                        ;version history
                        ;
                        ; V01-00       15-SEP-82
                        ;       1) Use Kernel PAR set for bootstrap temporary storage.
                        ;       2) # of blocks loaded by boot is now 8 vs. 12
                        ;       3) New Fault codes
                        ;       4) New ROM entries added: Receive Packet, Send Packet
                        ;       5) Don't set NXM vector until memory test
                        ;       6) Added missing branch to MFPS test
                        ;       7) Turn off MMU before using PAR's for storage
                        ;       8) Modified Fault lamp reporter. Now don't monitor Fault switch
                        ;          after it is pressed once.
                        ;       9) Added conditional 'SECNDP' to allow assembling a P.xxx
                        ;          (Secondary P) version of the bootstrap ROM:
                        ;              a) Set 'Select P' bits to 11(2).
                        ;              b) Puts 'Secondary P' code in States Register
                        ;       10) Now do a RESET to clear any pending K interrupts.
                        ;       11) Minor code compression.
                        ;
                        ;*****************************************************************
                        ;-
```

```
                        .sbttl  Useful Information

                ;+
                ;*****************************************************************************
                ;
                ; 1) MAXIMUM PROGRAM SIZE - This program must fit into a 2,048 byte ROM.
                ;          (Locations 17773000 thru 17776777)
                ;
                ; 2) Special Assembly Files Required - This program must be assembled
                ;       with the file 'MBCSPL.MAC/M' to supply I/O page addresses.
                ;
                ; 3) Special use of MMU PAR's (page address registers) - MMU  PAR's
                ;       are used as a convenient place to store information that
                ;       must be passed along to the last P.ioc test which succeeds the boot,
                ;       and are used as test locations. The contents of the USER PAR set are
                ;       preserved.  This is where boot reasons and failing addresses are stored.
                ;
                ;     PAR usage summary:
                ;           USER                KERNEL
                ;     0     preserved           TEST #
                ;     1     preserved           initial stack
                ;     2     preserved           initial stack
                ;     3     preserved           initial stack
                ;     4     preserved           initial stack/register bit tests
                ;     5     preserved           unit # & type of boot (R3 on entry)
                ;     6     preserved           CSR of TU58 to use (R4 on entry)
                ;     7     preserved           Validation of unit/type/CSR
                ;
                ; 4) Warm Boot entry restrictions: When using the 'WARM' boot feature,
                ;     the ROM must be entered with interrupts disabled (PRI=7), with
                ;     the MMU off, or in Kernel mode with the I/O page mapped.
                ;
                ; 5) Selection of load device - To force this program to select a particular
                ;       TU58 controller and unit for booting purposes, enter the ROM with the
                ;       following parameters:
                ;           a) R3 (lower byte) = Unit # of TU58 to be used for boot.
                ;                 (NOTE: unit byte may only be '0' or '1')
                ;              (upper byte) = Reserved, must be zero
                ;           b) R4 (16 bits)    = CSR address of TU58 to use for boot
                ;           c) R5 (16 bits)    = Validation of unit and CSR (R3 XOR R4)
                ;        If the data in R3 and R4 is valid (R3 XOR R4 = R5) then the boot will
                ;        first be attempted from the specified device.  If this load attempt
                ;        fails, or if the data in R3 and R4 is not valid, then the default
                ;        sequence of TU58 selection will be used. (see label 't58tst' for a
                ;        definition of the default TU58 selection sequence.)
                ;
                ; 6) Processor-time-dependent loops - All instruction sequences labeled 'PTIMEn'
                ;     are timing loops that depend on the basic cycle time of the F-11 CPU chip.
                ;
                ; 7) ROM VERSION NUMBER - The Version and Edit numbers of the ROM software are
                ;     stored in the last word of the ROM (address 176776).  The low byte is the
                ;     version #, the high byte is the edit #.
                ; 8) SECONDARY P VERSION - define 'SECNDP' to produce a Secondary P boot.
                ;
                ;*****************************************************************************
                ;-
```

```
                        .sbttl  Useful Information (continued)
                ;+
                ;*****************************************************************************
                ;
                ;       8) TU58 ERROR CODE TABLE - This program constructs an error table which
                ;          remembers why a particular TU58 unit could not be used for booting.
                ;          If the bootstrap fails, and the error code displayed in the fault
                ;          lamps is 'TU58 Failure', the error code table can be examined with
                ;          'F-11 ODT' to determine why none of the TU58's were suitable for booting.
                ;          The error code table begins at address 000400 in program memory and
                ;          is laid out as follows:
                ;              Address   Controller RCSR        Unit #
                ;              000400      172520               0
                ;              000402      172520               1
                ;              000404      172530               0
                ;              000406      172530               1
                ;              000410      172560               0
                ;              000412      172560               1
                ;
                ;              The low byte of each word in the table contains a code which
                ;              indicates the reason why the bootstrap software rejected that
                ;              particular TU58 unit. (The failure codes are defined in the
                ;              'Bit Definitions' section of this document).  The upper byte
                ;              of each word in the table contains the error reason from the
                ;              last end packet received from the TU58 (if any).  Note that if
                ;              a TU58 controller fails to synchronize, or fails its self-test,
                ;              the software failure code will be stored in unit 0's word of the
                ;              error table entry, and unit 1 will not be tried.
                ;
                ;*****************************************************************************
                ;-
```

```
                        .sbttl  Program Overview
                ;+
                ;*****************************************************************************
                ;       Test the hardcore of the P.ioc module, including the F-11,
                ;       Program Memory ( 9 Kwords), and a TU58.
                ;
                ;Inputs: 1) Unit # of TU58 to be used for boot. (R3, lower byte)
                ;        2) CSR of TU58 to be used for boot (R4)
                ;        3) Validation word for unit and CSR (R5)
                ;        4) type of boot (enter at 173000 for cold boot,
                ;           173006 for warm boot.
                ;
                ;Process: This test is initated each time the P.ioc subsystem is booted
                ;       or rebooted. Different entry points to the ROM allow the software
                ;       to determine whether the subsystem was being booted or rebooted
                ;       Testing begins with the ISP of the F-11 processor.  All basic
                ;       PDP-11 instructions, and all eight addressing modes are tested
                ;       Following the F-11 ISP tests, 9 Kwords of the P.ioc Program memory
                ;       are tested for proper addressing and ability to hold data and
                ;       proper parity.  The final test locates a working TU58 tape drive
                ;       that contains a MSCS8 System Cassette, and reads the first 8 blocks
                ;       from the tape to the 8 Kword partition in the Program memory.
                ;
                ;Outputs:(to 'Init P.ioc Test')
                ;
                ;       case 1 - F-11 test passes,8 Kword partition ok, TU58 ok
                ;               R0 = unit # of TU58 used for boot (lower byte)
                ;                    type of boot (upper bit of upper byte) 0=cold,1=warm
                ;               R1 = CSR address of TU58 used for boot
                ;               R2 = Base address of tested 8 Kword partition
                ;               'INIT' lamp is on.
                ;
                ;       case 2 - F-11 test fails
                ;               No parameters passed.  F-11 'hangs' immediately. (branch self)
                ;               'Init' lamp does not light, 'fault' lamp off.
                ;
                ;       case 3 - No working 8 Kword partition found in Program Memory
                ;                (or first 2048 bytes of Program Memory are faulty)
                ;               'Init' lamp is lighted
                ;               'Fault' lamp is lighted
                ;               Depressing fault switch causes blinking fault code display
                ;               'Fault code' = (memory test failure)
                ;               MSCS8 must be rebooted to attempt recovery
                ;               Program Memory module probably needs replacing
                ;
                ;       case 4 - No working TU58 found with bootable media mounted
                ;               'Init' lamp is lighted
                ;               'Fault' lamp is lighted
                ;               Depressing fault switch causes blinking fault code display
                ;               'Fault code' = (TU58 test failure)
                ;               MSCS8 must be rebooted to attempt recovery
                ;               No TU58 has system cassette mounted, or TU58's are faulty
                ;               (TU58 error code table contains reason why each TU58 unit
                ;                failed. See 'TU58 Error Table' under 'USEFUL INFORMATION'(above)
                ;
                ;*****************************************************************************
                ;-
```

```
                        .sbttl  Definitions
                        .sbttl  Address Definitions

                ;+
                ;*****************************************************************************
                ;
                ;       Address definitions
                ;
                ;*****************************************************************************
                ;-
        173000          basadd=173000           ;first byte in ROM
        176777          lasadd=176777           ;last byte in ROM
        172340          KUPAR0= 172340          ;address of kernel PAR 0
        172340          TESTNO= KUPAR0          ;Kernel PAR 0 is used to hold the test number

                        ;+
                        ;       TU58 ERROR TABLES
                        ;
                        ;       There are two words for each possible TU58 controller in this
                        ;       table.  In each of the two-word groups the first word is for
                        ;       unit 0 of that controller, the second word for unit one.
                        ;
                        ;       The low byte of each word in the table is for remembering the
                        ;       software reason that a particular TU58 unit was rejected.  The high
                        ;       byte of each word in the table, will hold the TU58 error code from
                        ;       a radial-serial-protocol end packet (if any).  A zero in either
                        ;       byte indicates no specific error was identified.
                        ;-

                ;low byte = ROM software error code, upper byte = end packet failure code (if an
        000400          SL1ER0= 400             ;TU58 01, unit 0
        000402          SL1ER1= SL1ER0+2        ;TU58 01, unit 1
        000404          SL2ER0= SL1ER1+2        ;TU58 02, unit 0
        000406          SL2ER1= SL2ER0+2        ;TU58 02, unit 1
        000410          SL0ER0= SL2ER1+2        ;console, unit 0
        000412          SL0ER1= SL0ER0+2        ;console, unit 1

                ;miscellaneous temporary R/W locations
        000414          CKSMEX= SL0ER1+2        ;expected radial serial packet checksum
        000416          CKSMAC= CKSMEX+2        ;actual radial serial packet checksum from TU58
        000420          TUERR0= CKSMAC+2        ;high byte = end packet code, low = software error code
        000420                  TUSWER= TUERR0  ;software error code byte
        000421                  TUHWER= TUERR0+1 ;hardware error code from end packet
        000422          TUERPT= TUERR0+2        ;ptr to current word of TU58 error table
000000                  .asect
        173000          .=basadd
```

```
                        .sbttl  Bit Definitions

                        ;+
                        ;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
                        ;
                        ;       Bit definitions
                        ;
                        ;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
                        ;-
                                ;+
                                ;--------------------------------------------------------------------
                                ;
                                ;       Miscellaneous definitions
                                ;
                                ;--------------------------------------------------------------------
                                ;-

000110                          TSDTMO= 72.     ;TU58 read response timer value( 72 times .55 sec = 40 sec)
040000                          PARSIZ= 16384.  ;size of program memory partition to be tested in bytes.
004000                          VECSIZ= 2048.   ;size of vector area to be tested in bytes.
000010                          LODSIZ= 8.      ;# of blocks to load for boot

                                ;+
                                ;--------------------------------------------------------------------
                                ;
                                ;       Fault register code definitions
                                ;
                                ;--------------------------------------------------------------------
                                ;-

000021                          prefal=21       ;P.ioc module failure
000022                          memfal=22       ;Memory module failure
000023                          tu58fl=23       ;TU58 failure code

                                ;+
                                ;--------------------------------------------------------------------
                                ;
                                ;       TU58 code definitions
                                ;
                                ;--------------------------------------------------------------------
                                ;-

                                ;Flag Byte Codes
000004                          init=4                  ;TU58 Initialize character
000002                          cmnd=2                  ;TU58 Command identifier
000001                          data=1                  ;TU58 Data identifier
000020                          cont=20                 ;TU58 Continue character
                                ;Op-codes
000002                          read=2                  ;read command
000003                          write=3                 ;write command
000005                          posit=5                 ;position tape command
000007                          diagno=7                ;diagnose command
000100                          endp=100                ;end packet op code
000001                          $TUBLK=1        ;NOTE: no longer defined in MSCDEF (14-aug-80)
```

```
391
392                                     ;+
393                                     ;-------------------------------------------------------------
394                                     ;
395                                     ;       P.ioc Control and Status Register Bit Definitions
396                                     ;
397                                     ;-------------------------------------------------------------
398                                     ;-
399
400     100000                          TERMEN=100000   ;1 when terminal enable, 0 when Secure
401     004000                          SUPERB=004000   ;1 when Swap memory boards
402     002000                          SUPBNK=002000   ;1 when Swap memory banks
403     001000                          SELP1 =001000   ;Bit 1 of P.ioc interrupt code
404     000400                          SELP0 =000400   ;Bit 0 of P.ioc interrupt code
405     000200                          ENXMEM=000200   ;1 when enable cmem arbitration
406     000100                          CLRBIT=000100   ;1 when clear I/O page device registers
407     000040                          HIPTST=000040   ;1 when enable high byte parity test
408     000020                          LOPTST=000020   ;1 when enable low byte parity test
409     000010                          LEDBIT=000010   ;1 when light STATE LED
410     000004                          NPRBIT=000004   ;1 when create a non-memory-access cycle
411     000002                          CNTIE =000002   ;1 when control memory interrupts enabled
412     000001                          LCKBIT=000001   ;1 when create a control memory lock cycle
413
414                                     ;+
415                                     ;-------------------------------------------------------------
416                                     ;
417                                     ;       TU58 Software Error codes
418                                     ;       (used in low byte of TU58 error table words)
419                                     ;
420                                     ;-------------------------------------------------------------
421                                     ;-
422     000001                          TUNXM=  1       ; 1 = NXM trap while accessing TU58 CSR's
423     000002                          TUSYNC= 2       ; 2 = TU58 synchronization sequence failed
424     000003                          TUSLFT= 3       ; 3 = TU58 self-test failed
425     000004                          TUSOFT= 4       ; 4 = TU58 unit did not contain bootable image
426     000005                          TUCKSM= 5       ; 5 = Checksum error in (received) radial-serial-packet
427     000006                          TUXMTO= 6       ; 6 = timeout on XMIT RDY when xmitting to TU58
428     000007                          TURCTO= 7       ; 7 = timeout on RCVR DONE when receiving from TU58
429     000010                          TURCER= 10      ;10 = UART error on received character (overrun, framing error
430     000011                          TUPROT= 11      ;11 = radial-serial-packet protcol error (unknown packet type)
431
432                                     ;+
433                                     ;-------------------------------------------------------------
434                                     ;       Special local definition of PDP-11 RESET instruction
435                                     ;       (Required since RESET is illegal in normal MSCS0 code,
436                                     ;       and MSCS0 Macro Library defines RESET to cause an error)
437                                     ;
438                                     ;-------------------------------------------------------------
439                                     ;-
440     000005                          RESETS= 000005  ;define our own mnemonic for a RESET instruction
```

```
                                ;+
                                ;---------------------------------------------------------------
                                ;
                                ;       P.ioc Switch/Display Register Bit Definitions
                                ;
                                ;---------------------------------------------------------------
                                ;-

                                FLTSW =004000   ;1 when 'Fault' switch depressed
                                BLKSW =002000   ;1 when 'Online' switch depressed
                                HBSW  =001000   ;1 when 'First Unmarked' switch depressed
                                HASW  =000400   ;1 when 'Second Unmarked' switch depressed
                                INILMP=000020   ;1 when light 'Init' lamp
                                FLTLMP=000010   ;1 when light 'Fault' lamp
                                BLKLMP=000004   ;1 when light 'Online' lamp
                                HBLMP =000002   ;1 when light 'First Unmarked' lamp
                                HALMP =000001   ;1 when light 'Second Unmarked' lamp
                                ALLLMP=HALMP!HBLMP!BLKLMP!FLTLMP!INILMP ;Inclusive OR of all lamp bits
```

```
                        .sbttl  Alternate Entry Points and Pointer Table


;+
;******************************************************************************
;
;       Alternate entry points to this program are supplied to allow
;       the distinction between a 'cold' boot and a 'warm' boot.
;       (A cold boot is the result of power-up, or '173006' to ODT.  No
;       saved context exists for a cold boot.  A warm boot indicates that
;       there is saved context in the User PAR set, to be used by the Init
;       P.ioc test.)
;
;Inputs:
;               R3 = unit # of TU58 to use for boot (lower byte)
;               R4 = CSR address of TU58 to use for boot (16 bits)
;               R5 = Validation of unit/CSR (R3 XOR R4)
;               For Warm boots, the User PAR set contains the reason
;               for the reboot, and other context.  This context is not
;               used by the Bootstrap ROM, it is merely not disturbed.
;
;process:       Set up a 'type of boot' parameter, depending upon which
;               entry into the ROM is used.  The following entry points
;               are provided:
;                       1) Entry for 'Cold' boot
;                       2) Entry for 'Warm' boot
;
;outputs:       R3, R4, and R5 preserved
;               'Type of boot' encoded in upper  bit of R3
;
;               The value of the upper  bit of R3 is as follows:
;                       cold boot = 0
;                       warm boot = 1
;
;notes:
;
;History of changes to this routine
;
;******************************************************************************
;-
```

```
                        coldbt: ;+
                                ;-------------------------------------------------------------
                                ;
                                ;       Entry for 'Cold' boot
                                ;
                                ;-------------------------------------------------------------
                                ;-

173000  106427  000340              mtps    #340    ;prevents real time clock from interrupting
173004  000137  173032              jmp     F11TST  ;cold boot entry


                                ;+
                                ;-------------------------------------------------------------
                                ;
                                ;       Entry for 'Warm' boot   (base address + 10)
                                ;
                                ;-------------------------------------------------------------
                                ;-

173010  052703  100000  BTYPE2: bis     #100000,R3      ;set R3 negative to indicate warm boot
173014  000410                  br      ALTENT          ;start bootstrap tests


                                ;+
                                ;-------------------------------------------------------------
                                ;
                                ;       Table of pointers to ROM routines that are used from RAM
                                ;       (by the INIT P.10C test)
                                ;
                                ;       The relative position of these pointers, with respect to the
                                ;       starting address of the ROM, must NEVER be changed.
                                ;
                                ;-------------------------------------------------------------
                                ;-

173016  175630  TU58SY: .word  T58SYC  ;base + 16      TU58 synchronization routine
173020  175442  TU58RD: .word  T58READ ;base + 20      TU58 read routine
173022  176136  TU58SN: .word  SENDPK  ;base + 22      Send TU58 command packet
173024  176570  MIMENT: .word  MOVINV  ;base + 24      Moving Inversions RAM test
173026  174572  ERRPT:  .word  ANYFAL  ;base + 26      Fault code display routine
173030  176242  TU58RC: .word  RECVPK  ;base + 30      Receive TU58 command or data packet

                        ;****************** END OF POINTER TABLE **********************
```

```
                        .sbttl  ..TEST 0 - F-11 ISP Test
                ;+
                ;*****************************************************************************
                ;       Test the basic PDP-11 instruction set and addressing modes.
                ;
                ;Inputs:        PSW Priority field = 7 (interrupts disabled) upon entry
                ;               R3 = Unit # to use for boot (lower byte, value of 0 or 1 only)
                ;               R4 = CSR address of TU58 to use for boot (16 bit I/O page address)
                ;               R5 = Validation for R3 and R4 (R3 XOR R4)
                ;                  (validation prevents spurious device selection)
                ;
                ;Process:  Test the Instruction Set Processor (ISP) of the P.ioc's F-11
                ;         processor.
                ;       Instructions tested:
                ;               CLR(B),COM(B),INC(B),DEC(B),NEG(B),TST(B),ROR(B),ROL(B),
                ;               ASR(B),ASL(B),SWAB,MOP,ADC(B),CCC,CLN,CLV,CLZ,SCC,SEN,SEV,SEZ
                ;
                ;               MOV(B),CMP(B),BIT(B),BIC(B),BIS(B),ADD,SUB
                ;
                ;               BR,BNE,BEQ,BPL,BMI,BCC(BHIS),BCS(BLO),BGE
                ;               BLT,BGT,BLE,BHI,BLOS,BVC,BVS
                ;
                ;               JMP,JSR,RTS,SOB,MTPS,MFPS
                ;
                ;       Addressing modes tested:
                ;               All eight addressing modes are tested
                ;
                ;       Instructions NOT tested:
                ;               ASH,ASHC,BPT,DIV,EMT,HALT
                ;               IOT,MARK,MFPI,MTPI,MUL,RESET,RTI,RTT,SBC
                ;               SPL,TRAP,WAIT,XOR
                ;
                ;       F-11 Logic NOT tested:
                ;               Memory Mapping, Interrupt priority and arbitration,
                ;               Stack Overflow, Power Fail, Bus Time-out Logic
                ;
                ;       P.ioc logic NOT tested
                ;               Control Memory Windows, ECC tables
                ;               Operator console interface, 'K' status registers
                ;               Control and arbitration of Data and Control buses
                ;               Remainder of Program Memory (total size minus 8 Kwords)
                ;
                ;Outputs:
                ;       Case 1 - no failure detected
                ;               'Init' lamp is lighted
                ;               Control passed to Program Memory Test
                ;               R3,R4, and R5 are preserved in MMU Kernel PAR's 5,6, and 7
                ;
                ;       Case 2 - failure detected
                ;               F-11 CPU is 'hung'.  PC of 'hung' identifies failure.
                ;               'Init' lamp is out, 'Fault' lamp is out
                ;
                ;Notes:
                ;History of changes to this test:
                ;*****************************************************************************
                ;-
```

```
                                F11TST: ;+
                                        ;-------------------------------------------------------------
                                        ;
                                        ;       Initialize stack pointer such that if a trap or interrupt
                                        ;       occurs, the F-11 will generate a double bus error
                                        ;
                                        ;-------------------------------------------------------------
                                        ;-
 614 173032  042703  100000             bic     #100000,R3      ;set 'type of boot' to 'cold boot'
                                ALTENT: RESETS                  ;clear all DCP lamps, clear all K interrupts
 615 173036  005037  172340             clr     TESTNO          ;set test # to 0
 616 173040  005006                     clr     SP              ;SP gets a 0
 617 173044  005106                     com     SP              ;SP gets 177777, causes DBL BUS ERR if trap occurs
                                        ;+
                                        ;-------------------------------------------------------------
                                        ;
                                        ;       Test Branch and Condition Code Instructions
                                        ;
                                        ;       Test BR, BNE, BEQ, BPL, BMI, BCC(BHIS), BCS(BLO)
                                        ;            BGE, BLT, BGT, BLE, BHI, BLOS, CCC, SCC
                                        ;            SEN, BVS, BVC, SEN, CLN, SEZ, CLZ, SEC, CLC
                                        ;
                                        ;-------------------------------------------------------------
                                        ;-
 648 173050  000277                     scc                     ; N (- 1, Z (- 1, V (- 1, C (- 1
 649 173052  001377                     bne     .               ;nbe (no branch expected)
 650 173054  100377                     bpl     .               ;nbe
 651 173056  103377                     bcc     .               ;nbe
 652 173060  003377                     bgt     .               ;nbe
 653 173062  101377                     bhi     .               ;nbe
 654 173064  002777                     blt     .               ;nbe
 655 173066  102377                     bvc     .               ;nbe
 656 173070  000250                     cln                     ;n (- 0
 657 173072  002377                     bge     .               ;nbe (n clear, v set)
 658 173074  000270                     sen                     ;n (- 1
 659 173076  002777                     blt     .               ;nbe (n set, v set)
 660 173100  000241                     clc                     ;c (- 0
 661 173102  103777                     bcs     .               ;nbe
 662 173104  000261                     sec                     ;c (- 1
 663 173106  103377                     bcc     .               ;nbe
 664 173110  000244                     clz                     ;z (- 0
 665 173112  001777                     beq     .               ;nbe
 666 173114  000264                     sez                     ;z (- 1
 667 173116  001377                     bne     .               ;nbe
 668 173120  000242                     clv                     ;v (- 0
 669 173122  102777                     bvs     .               ;nbe
 670 173124  000262                     sev                     ;v (- 1
 671 173126  102377                     bvc     .               ;nbe
 672 173130  000257                     ccc                     ; N (- 0, Z (- 0, V (- 0, C (- 0
 673 173132  001777                     beq     .               ;nbe (no branch expected)
 674 173134  100777                     bmi     .               ;nbe
 675 173136  103777                     bcs     .               ;nbe
 676 173140  002777                     blt     .               ;nbe
 677 173142  003777                     ble     .               ;nbe
 678 173144  101777                     blos    .               ;nbe
 679 173146  102777                     bvs     .               ;nbe
```

```
                                        br      10              ;branch forward
                                        br      .               ;failed to branch forward
                                10:     sen                     ;set the N bit ( N (- 1 )
                                        bpl     .               ;nbe (no branch expected)
                                        bge     .               ;nbe
                                        bgt     .               ;nbe
                                        blt     20              ;should branch
                                        br      .               ;previous branch failed
                                30:     ble     100             ;should branch
                                20:     bmi     30              ;should branch backwards
                                        br      .               ;backwards branch failed

                                100:    ;+
                                        ;-------------------------------------------------------------------
                                        ;
                                        ;       Test CLR, COM, INC, DEC, NEG, TST, ROR, ROL, ASR, ASL, SWAB, NOP
                                        ;          and SEC
                                        ;
                                        ;       Address mode 0, using R0
                                        ;
                                        ;-------------------------------------------------------------------
                                        ;-

                                        clr     R0              ; R0 (- 0,      C (- 0
                                        com     R0              ; R0 (- 177777, C (- 1
                                        inc     R0              ; R0 (- 000000, C = 1 (C not affected by inc)
                                        dec     R0              ; R0 (- 177777, C = 1 (C not affected by dec)
                                        nop                     ;insure nop does not interfere
                                        bcc     .               ;hang here if c bit is clear, or bcc fails
                                        neg     R0              ; R0 (- 000001, C (- 1
                                        bcc     .               ;hang here if c bit is clear, or bcc fails
                                        tst     R0              ; R0 = 000001,  C (- 0
                                        beq     .               ;hang here if R0 = 0, or tst or beq fails
                                        ror     R0              ; R0 (- 000000, C (- 1
                                        bne     .               ;hang here if R0 not 0
                                        bcc     .               ;hang here if C didn't get loaded with a 1
                                        rol     R0              ; R0 (- 000001, C (- 0
                                        nop                     ;insure nop does not interfere
                                        bcs     .               ;hang if c bit is set
                                        beq     .               ;hang if R0 is 0
                                        sec                     ; R0 = 000001, c (- 1
                                        swab    R0              ; R0 (- 000400, C (- 0
                                        bcs     .               ;hang if swab did not clear c bit
                                        tstb    R0              ;test for lower byte of R0 = 0
                                        bne     .               ;hang here if tstb failed, or R0 (lower) ne 0
                                        swab    R0              ; R0 (- 000001, C (- 0
                                        tstb    R0              ;test that lower byte of R0 now not zero
                                        beq     .               ;hang if tstb failed, or R0 (lower) = 0
```

```
                        ;+
                        ;-------------------------------------------------------------
                        ;
                        ;       Test MOV, CMP, BIT, BIC, BIS, ADD, SUB
                        ;
                        ;       Address mode 0, using R0, R1, and SP (R6)
                        ;
                        ;-------------------------------------------------------------
                        ;-

173260  010600          mov     SP, R0          ; R0 <- 177777
173262  020600          cmp     SP,R0           ; R0 = 177777, SP = 177777
173264  001377          bne     .               ;hang if SP and R0 not equal
173266  040600          bic     SP,R0           ; R0 <- 000000
173270  001377          bne     .               ;hang if R0 not 0
173272  020600          cmp     SP,R0           ; R0 = 000000, SP = 177777
173274  001777          beq     .               ;hang if compare failed
173276  005200          inc     R0              ; R0 <- 000001
173300  060600          add     SP,R0           ; R0 <- 000000, C <- 1
173302  001377          bne     .               ;hang if add failed
173304  103377          bcc     .               ;hang if c bit did set
173306  050600          bis     SP,R0           ; R0 <- 177777
173310  005100          com     R0              ; R0 <- 000000
173312  001377          bne     .               ;hang if R0 not zero
173314  160600          sub     SP,R0           ; R0 <- 000001
173316  001777          beq     .               ;hang if R0 = 0
173320  005300          dec     R0              ; R0 <- 000000
173322  001377          bne     .               ;hang if R0 not 0
173324  005200          inc     R0              ; R0 <- 000001
173326  005001          clr     R1              ; R1 <- 000000
173330  005201          inc     R1              ; R1 <- 000001
173332  040001          bic     R0,R1           ; R1 <- 000000
173334  001377          bne     .               ;hang if R1 not 0
173336  005201          inc     R1              ; R1 <- 000001
173340  005100          com     R0              ; R0 <- 177776
173342  060100          add     R1,R0           ; R0 <- 177777,  C <- 0
173344  103777          bcs     .               ;hang if c bit is set
173346  005100          com     R0              ; R0 <- 000000
173350  001377          bne     .               ;hang if R0 not 0
```

```
                        ;+
                        ;-------------------------------------------------------------
                        ;
                        ;       Test CLRB, COMB, INCB, DECB, NEGB, TSTB, RORB, ROLB
                        ;            ASRB, ASLB, MOVB, CMPB, BITB, BICB, BISB
                        ;
                        ;       Address mode 0. .sing R0 and R1
                        ;
                        ;-------------------------------------------------------------
                        ;-

173352  005100          com     R0              ; R0 (- 177777
173354  110001          movb    R0,R1           ; R1 (- 177777 (movb to register sign extends)
173356  005101          com     R1              ; R1 (- 000000
173360  001377          bne     .               ;hang if R1 not 0
173362  110100          movb    R1,R0           ; R0 (- 000000 (movb to register sign extends)
173364  001377          bne     .               ;hang if R0 not 0
173366  105100          comb    R0              ; R0 (- 000377
173370  105700          tstb    R0              ; R0 = 000377
173372  001777          beq     .               ;hang if lower byte is zero
173374  000300          swab    R0              ; R0 (- 177400
173376  105700          tstb    R0              ; R0 = 177400 (lower byte = 0)
173400  001377          bne     .               ;hang if lower byte of R0 not 0
173402  005101          com     R1              ; R1 (- 177777
173404  105001          clrb    R1              ; R1 (- 177400
173406  105701          tstb    R1              ; R1 = 177400
173410  001377          bne     .               ;hang if lower byte of R1 not 0
173412  000301          swab    R1              ; R1 (- 000377
173414  105701          tstb    R1              ; R1 = 000377
173416  001777          beq     .               ;hang if lower byte of R1 = 0
173420  120001          cmpb    R0,R1           ; R0 = 177400, R1 = 000377
173422  001777          beq     .               ;hang if lower bytes are equal
173424  000300          swab    R0              ; R0 (- 000377
173426  106300          aslb    R0              ; R0 (- 000376, C (- 1
173430  103377          bcc     .               ;hang if c bit is not set
173432  120001          cmpb    R0,R1           ; R0 (lower) = 376, R1 (lower) = 377
173434  001777          beq     .               ;hang if lower bytes are equal
173436  105200          incb    R0              ; R0 (- 000377
173440  006300          asl     R0              ; R0 (- 000776
173442  105200          incb    R0              ; R0 (- 000777
173444  120001          cmpb    R0,R1           ; R0 = 000777, R1 = 000377 (equal in lower byte)
173446  001377          bne     :               ;hang if lower bytes are not equal
173450  105300          decb    R0              ; R0 (- 000776
173452  106200          asrb    R0              ; R0 (- 000777 (lower byte sign extends)
173454  120001          cmpb    R0,R1           ; R0 = 000777, R1 = 000377 (equal in lower byte)
173456  001377          bne     .               ;hang if lower bytes are not equal
173460  140100          bicb    R1,R0           ; R0 (- 000400 (lower byte cleared)
173462  001377          bne     .               ;hang if lower byte did not result in a 0
173464  150100          bisb    R1,R0           ; R0 (- 000777
173466  105100          comb    R0              ; R0 (- 000400 (lower byte 0)
173470  001377          bne     .               ;hang if lower byte not 0
173472  130100          bitb    R1,R0           ; R0 = 000400, R1 = 000377
173474  001377          bne     .               ;hang if bit test result was a 1
173476  105200          incb    R0              ; R0 (- 000401
173500  105101          comb    R1              ; R1 (- 000000
173502  105201          incb    R1              ; R1 (- 000001
173504  130100          bitb    R1,R0           ; R0 = 000401, R1 = 000001
```

```
020 173506  001777                 beq     .               ;hang if bit test result was zero
029 173510  000241                 clc                     ; C (- 0
030 173512  106001                 rorb    R1              ; R1 (- 000000, C (- 1
031 173514  103377                 bcc     .               ;hang if c bit is not a 1
032 173516  001377                 bne     .               ;hang if result in R1 not 0
033 173520  106101                 rolb    R1              ; R1 (- 000001, C (- 0
034 173522  103777                 bcs     .               ;hang if c bit is set
035 173524  001777                 beq     .               ;hang if result in R1 is 0
036 173526  105401                 negb    R1              ; R1 (- 000377, c (- 1
037 173530  105201                 incb    R1              ; R1 (- 000000, C remains 1 (not affected)
038 173532  103377                 bcc     .               ;hang if c bit clear
039 173534  001377                 bne     .               ;hang if result in R1 is not zero
040 173536  000300                 swab    R0              ; R0 (- 000401
041 173540  105300                 decb    R0              ; R0 (- 000400
042 173542  001377                 bne     .               ;hang if result in R0 (lower) is not 0
043 173544  005700                 tst     R0              ; R0 = 000400
044 173546  001777                 beq     .               ;hang if result in R0 is 0
045
046
047                                ;+
048                                ;-------------------------------------------------------------
049                                ;
050                                ;       Test JMP and SOB instructions
051                                ;
052                                ;       Using R0, R1, and R2 (jmp uses mode 6 PC addressing)
053                                ;
054                                ;-------------------------------------------------------------
055                                ;-
056
057 173550  005000                 clr     R0              ; R0 (- 000000
058 173552  005001                 clr     R1              ; R1 (- 000000
059 173554  005002                 clr     R2              ; R2 (- 000000
060 173556  005200                 inc     R0              ; R0 (- 000001
061 173560  005201                 inc     R1              ; R1 (- 000001
062 173562  006300                 asl     R0              ; R0 (- 000002
063 173564  006301                 asl     R1              ; R1 (- 000002
064 173566  105301         sobbak: decb    R1              ; R1 (- R1 minus 1
065 173570  105202                 incb    R2              ; R2 (- R2 plus 1
066 173572  077003                 sob     R0,sobbak       ; R0 (- R0 minus 1, branch if result not 0
067 173574  105701                 tstb    R1              ; R1 should now be zero also
068 173576  001377                 bne     .               ;hang if R1 is not 0
069 173600  105302                 decb    R2              ; R2 (- 1
070 173602  105302                 decb    R2              ; R2 (- 0
071 173604  001377                 bne     .               ;hang if R2 did not equal 2 after sob loop above
072 173606  000137  173614         jmp     jmptar          ;test a forward jump
073 173612  000777                 br      .               ;hang if jump failed
074 173614  000137  173630 jmptar: jmp     jmptr1          ;test another forward jump
075 173620  000777                 br      .               ;hang if jump failed
076 173622  000137  173636 jmptr2: jmp     F11ADD          ;test another forward jump
077 173626  000777                 br      .               ;hang if jump failed
078 173630  000137  173622 jmptr1: jmp     jmptr2          ;test a backward jump
079 173634  000777                 br      .               ;hang if jump failed
```

```
                        F11ADD: ;+
                                ;-------------------------------------------------------------
                                ;
                                ;       Test addressing modes
                                ;
                                ;       Address modes 1, 2, 3, 4, 5, 6, and 7 are tested
                                ;
                                ;       R0 and R1 are used
                                ;
                                ;-------------------------------------------------------------
                                ;-

173636  012700  173724          mov     #moddat,R0      ;R0 gets pointer to data for this test  (SM2)
173642  011001                  mov     (R0),R1         ;R1 gets test data                      (SM1)
173644  022001                  cmp     (R0)+,R1        ;check data against itself              (SM2)
173646  001377                  bne     .               ;hang if datum not equal
173650  020027  173726          cmp     R0,#MODDAT+2    ;check R0 for correct contents
173654  001377                  bne     .               ;hang if R0 is wrong
173656  063001                  add     @(R0)+,R1       ;add test data to R1                    (SM3)
173660  020027  173730          cmp     R0,#MODDAT+4    ;check R0 for correct value
173664  001377                  bne     .               ;hang if wrong value in R0
173666  163001                  sub     @-(R0),R1       ;subtract test data from R1             (SM5)
173670  043001                  bic     -(R0),R1        ;clear R1 by bit clear with same data   (SM4)
173672  001377                  bne     .               ;hang if R1 was not cleared
173674  020027  173724          cmp     R0,#MODDAT      ;check R0 for correct value
173700  001377                  bne     .               ;hang if R0 is incorrect
173702  056001  000004          bis     4(R0),R1        ; R1 <- 177777                          (SM6)
173706  037001  000006          bit     @6(R0),R1       ;test for any bits set                  (SM7)
173712  001777                  beq     .               ;hang if no bits are set
173714  005101                  com     R1              ; R1 <- 000000
173716  001377                  bne     .               ;hang if R1 was not all ones before com
173720  000405                  br      F11MTP          ;transfer to next test
173722  000777                  br      .               ;previous branch failed

173724  173724          moddat: .word   moddat          ;data locations for addressing mode test
173726  173724                  .word   moddat
173730  177777          moddt1: .word   177777
173732  173730                  .word   moddt1
```

```
920 173734                      F11MTP: ;+
921                                     ;-------------------------------------------------------
922                                     ;
923                                     ;       Test MFPS instruction
924                                     ;
925                                     ;       Address mode 0
926                                     ;
927                                     ;       Note: Until interrupt logic is tested, (in 'Inst P.iec test'
928                                     ;             following the ROM test), testing of the priority bits
929                                     ;             in the PSW is delayed to avoid uncontrolled detection
930                                     ;             of failures in the interrupt logic.  This test restricts
931                                     ;             itself to checking that the priority is now 7, and that
932                                     ;             the N, Z, V, and C bits can be set and cleared.
933                                     ;
934                                     ;-------------------------------------------------------
935                                     ;-
936
937 173734  000277                      scc                     ; N (- 1, Z (- 1, V (- 1, C (- 1
938 173736  106700                      mfps    R0              ; R0 (- 177757 (lower byte of PSW, sign extend)
939 173740  020027  177757              cmp     R0,#177757      ; check for pri 7, all cc bits set
940 173744  001377                      bne     .               ;hang if mfps failed
941 173746  000257                      ccc                     ; N (- 0, Z (- 0, V (- 0, C (- 0
942 173750  106700                      mfps    R0              ; R0 (- 177740
943 173752  020027  177740              cmp     R0,#177740      ; check for pri 7, all cc bits clear
944 173756  001377                      bne     .               ;hang if mfps failed
945                                     ;+
946                                     ;-------------------------------------------------------
947                                     ;
948                                     ;       Test JSR and RTS instructions
949                                     ;       TURN OFF THE MMU
950                                     ;
951                                     ;       NOTE: In order to test JSR, the stack must be used.  This
952                                     ;       requires using some location in memory for a one word stack.
953                                     ;
954                                     ;       This test uses part of the MMU's Kernel Page Address Register Set as
955                                     ;       a stack area.  (8 contiguous 16 bit registers)
956                                     ;
957                                     ;-------------------------------------------------------
958                                     ;-
959 173760  005037  0000006             clr     @#MMR3          ;TURN OFF THE MMU
960 173764  012706  0000006             mov     #KPAR5,SP       ;point SP to Kernel PAR5
961                                                             ;(KPAR's 4,3,2,1, and 0 used for stack)
962 173770  004737  174002              jsr     PC,JSRTST       ;test JSR instruction. Return at call46
963 173774  000777                      br      .               ;hang if JSR returns to wrong location
964 173776  000405                      br      F11ADC          ;Correct return from JSR above
965 174000  000777                      br      .               ;hang if JSR returned to wrong location
966
967 174002  062716  000002      JSRTST: add     #2,(SP)         ;update return address by 2
968 174006  000207                      rts     pc              ;test the RTS instruction
969 174010  000777                      br      .               ;hang if RTS failed to return
```

```
 971 174012                     F11ADC: ;+
 972                                    ;-------------------------------------------------------------
 973                                    ;
 974                                    ;       Test ADC and ADCB instructions
 975                                    ;
 976                                    ;       Move the boot parameters from R3, R4, and R5 to the Memory
 977                                    ;       Management Kernel PAR's 5, 6, and 7, where they will be
 978                                    ;       preserved across the memory and TU58 tests to follow.
 979                                    ;
 980                                    ;-------------------------------------------------------------
 981                                    ;-
 982
 983 174012  012701  0000006                mov     #KPAR5,R1       ;point R1 to Kernel par 5
 984 174016  010321                         mov     R3,(R1)+        ;save unit and type of boot in UPAR5
 985 174020  010421                         mov     R4,(R1)+        ;save CSR of TU58 in UPAR6
 986 174022  010511                         mov     R5,(R1)         ;save validation word for unit and CSR
 987 174024  005001                         clr     R1              ; R1 = 0, C = 0
 988 174026  005301                         adc     R1              ; R1 = 0        (0+0=0)
 989 174030  001113                         bne     CPUFAL          ;br if answer is non-zero
 990 174032  103512                         bcs     CPUFAL          ;br if c bit is set
 991 174034  105501                         adcb    R1              ; R1 = 0        (0+0=0)
 992 174036  001110                         bne     CPUFAL          ;br if answer is non-zero
 993 174040  005101                         com     R1              ; R1 = [illegible], C = 1
 994 174042  005301                         adc     R1              ; R1 = 0, C = 1 (177777+1=0)
 995 174044  001105                         bne     CPUFAL          ;br if answer is not zero
 996 174046  103104                         bcc     CPUFAL          ;br if c bit did not set
 997 174050  012702  177777                mov     #177777,R2      ; R2 = 177777, C = 1
 998 174054  105502                         adcb    R2              ; R2 = 177400, C = 1
 999 174056  103100                         bcc     CPUFAL          ;br if c bit did not set
1000 174060  020227  177400                cmp     R2,#177400      ;check for correct answer in R2
1001 174064  001075                         bne     CPUFAL          ;br if wrong answer
1002 174066  000261                         sec                     ; C = 1
1003 174070  005501                         adc     R1              ; R1 = 1, C = 0 (0+1=1)
1004 174072  103472                         bcs     CPUFAL          ;br if c bit is set
1005 174074  005301                         dec     R1              ; R1 = 0        (1-1=0)
1006 174076  001070                         bne     CPUFAL          ;br if R1 did not contain a 1
```

```
                                        ;+
                                        ;-------------------------------------------------------------
                                        ;
                                        ;       Perform bit tests of R1 thru R6
                                        ;
                                        ;-------------------------------------------------------------
                                        ;-

174100  012701  100000                  mov     #100000,R1      ;preset all registers to 100000
174104  010102                          mov     R1,R2
174106  010203                          mov     R2,R3
174110  010304                          mov     R3,R4
174112  010405                          mov     R4,R5
174114  010506                          mov     R5,SP
174116  001460                          beq     CPUFAL          ;abort if bit was dropped along the way
174120  010637  0000006                 mov     SP,@#KPAR4      ;set Kernel PAR4 to same pattern
174124  012700  000017                  mov     #15.,R0         ;R0 will count 15 shifts
174130  006201                  100:    asr     R1              ;shift each register 1 bit right, sign extending 1 s
174132  006202                          asr     R2
174134  006203                          asr     R3
174136  006204                          asr     R4
174140  006205                          asr     R5
174142  006206                          asr     SP
174144  006237  0000006                 asr     @#KPAR4
174150  020102                          cmp     R1,R2           ;check for new pattern, with one additional '1' bit
174152  001042                          bne     CPUFAL
174154  020304                          cmp     R3,R4
174156  001040                          bne     CPUFAL
174160  020506                          cmp     R5,SP
174162  001036                          bne     CPUFAL
174164  020637  0000006                 cmp     SP,@#KPAR4
174170  001033                          bne     CPUFAL
174172  077022                          sob     R0,100          ;branch until 15 shifts have been performed
174174  023727  0000006 177777          cmp     @#KPAR4,#177777 ;is final pattern all 1's ??
174202  001026                          bne     CPUFAL
```

```
                                        ;+
                                        ;-------------------------------------------------------------
                                        ;
                                        ;       Basic Adder Test
                                        ;
                                        ;-------------------------------------------------------------
                                        ;-

174204  012706  000006                  mov     #SKPAR5,SP      ;set stack pointer to MMU Kernel PAR5
174210  005001                          clr     R1              ; R1 = 0
174212  062701  125252                  add     #125252,R1      ; R1 = 125252, N=1, Z=0, V=0, C=0
174216  062701  052525                  add     #052525,R1      ; R1 = 177777, N=1, Z=0, V=0, C=0
174222  103416                          bcs     CPUFAL          ;br if c bit is set
174224  003015                          bgt     CPUFAL          ;br if n bit is not set
174226  102414                          bvs     CPUFAL          ;br if v bit is set
174230  062701  146314                  add     #146314,R1      ; R1 = 146313, N=1, Z=0, V=0, C=1
174234  103011                          bcc     CPUFAL          ;br if c bit is not set
174236  062701  125252                  add     #125252,R1      ; R1 = 073565, N=0, Z=0, V=1, C=1
174242  102006                          bvc     CPUFAL          ;br if v bit did not set
174244  103005                          bcc     CPUFAL          ;br if c bit did not set
174246  062701  104213                  add     #-73565,R1      ; R1 = 000000, N=0, Z=1, V=0, C=1
174252  001002                          bne     CPUFAL          ;br if R1 not zero
174254  102401                          bvs     CPUFAL          ;br if v bit is set
174256  103403                          bcs     F11DON          ;br if c bit is set, all F-11 tests have passed


174260                          CPUFAL: ;+
                                        ;-------------------------------------------------------------
                                        ;
                                        ;       F-11 CPU test failure. (only used for last few F-11 tests)
                                        ;
                                        ;       1) Set to R0 to fault code for 'P.ioc failure'
                                        ;
                                        ;       2) Light the FAULT lamp
                                        ;
                                        ;       3) Monitor the FAULT switch
                                        ;
                                        ;-------------------------------------------------------------
                                        ;-

174260  012700  000021                  mov     #PIOFAL,R0      ;R0 gets error code for P.ioc failure
174264  000542                          br      ANYFAL          ;go to common error display routine

174266                          F11DON: ;+
                                        ;-------------------------------------------------------------
                                        ;
                                        ;       Light the 'Init' lamp to show F-11 test passed
                                        ;
                                        ;-------------------------------------------------------------
                                        ;-

174266  052737  000020  0000006         bis     #INILMP,@#CSR   ;light the init lamp
```

```
1097                                    .sbttl  MEMORY TESTS
1098                                ;+
1099                                ;*****************************************************************************
1100                                ;
1101                                ;       Test the first 2,048 bytes of Program Memory, and
1102                                ;       Find a working 8 Kword partition in Program Memory
1103                                ;
1104                                ;inputs:        MMU Kernel PAR 5 = unit 0 (lower byte), type of boot (upper byte)
1105                                ;               MMU Kernel PAR 6 = CSR of TU58 to use for boot
1106                                ;               MMU Kernel PAR 7 = validation word for unit and CSR parameters
1107                                ;
1108                                ;process: Find an 8 Kword partition in Program Memory that passes a combined
1109                                ;       data integrity and addressing test.  In addition to this 8 Kword
1110                                ;       partition, the first 2048 bytes of Program memory must be working for
1111                                ;       use as the PDP-11 interrupt and trap vector area.
1112                                ;       If the first 2048 bytes do not work, the entire test
1113                                ;       is considered to have failed.  If the first 2048 bytes are working,
1114                                ;       then try to find any 8 Kword chunk of Program memory that works.  If the
1115                                ;       first 2048 bytes are working, but no contiguous 8 Kword partition can
1116                                ;       be found, the test fails.
1117                                ;
1118                                ;outputs:  case 1 - no failure detected
1119                                ;               'Init' lamp is on, 'Fault' lamp is off
1120                                ;               R0 = Base address of tested 8 Kword partition
1121                                ;               MMU Kernel PAR's 5,6, and 7 are unchanged
1122                                ;               Control is passed to the TU58 test
1123                                ;
1124                                ;           case 2 - failure detected (see above for definition of failure)
1125                                ;               'Fault' lamp is on
1126                                ;               'Init' lamp is on
1127                                ;               F-11 is in loop, monitoring the fault lamp
1128                                ;               Depressing the fault lamp causes blinking display of fault code.
1129                                ;               F-11 must be rebooted to attempt recovery
1130                                ;
1131                                ;notes:
1132                                ;       1) the variable 'PARSIZ' defines the size of the partition to be tested.
1133                                ;       2) the variable 'VECSIZ' defines the size of the vector area to test.
1134                                ;
1135                                ;history of changes to this test:
1136                                ;
1137                                ;*****************************************************************************
1138                                ;-
```

```
                                        .sbttl  ..TEST 1 - Program Memory (Swap Bits)
                                ;+
                                ;*****************************************************************************
                                ;
                                ;       Test the 'Swap Boards' and 'Swap Banks' bits in the P.ioc CSR
                                ;
                                ;Inputs:  none
                                ;
                                ;Process: 1) set both swap bits
                                ;         2) Check that 'swap boards' is set
                                ;         3) Clear the 'swap boards' bit
                                ;         4) Check that the 'swap banks' bit is still set
                                ;         5) Clear the 'swap banks' bit
                                ;         6) Check that both swap bits are clear
                                ;
                                ;Outputs: Declare a P.ioc failure if any of the tests fail.
                                ;
                                ;*****************************************************************************
                                ;-
1159 174274  005237  172340             inc     TESTNO          ;set test # to 1
1160 174300  012704  000006             mov     #@PIOCSR,R4     ;point R4 to P.ioc's CSR
1161 174304  052714  006000             bis     #SWPBRD!SWPBNK,(R4) ;set both swap bits
1162 174310  032714  004000             bit     #SWPBRD,(R4)    ;check the swap board bit
1163 174314  001003                     bne     5$              ;br if bit is set
1164 174316  042714  006000     4$:     bic     #SWPBRD!SWPBNK,(R4) ;try to clear both bits
1165 174322  000756                     br      CPUFAL          ;declare a P.ioc failure
1166
1167 174324  042714  004000     5$:     bic     #SWPBRD,(R4)    ;clear the swap boards bit
1168 174330  032714  006000             bit     #SWPBRD!SWPBNK,(R4) ;test both bits (one should be set)
1169 174334  001770                     beq     4$              ;br if neither bit is set
1170 174336  042714  002000             bic     #SWPBNK,(R4)    ;clear the swap banks bit (both clear now)
1171 174342  032714  006000             bit     #SWPBRD!SWPBNK,(R4) ;check for both bits clear
1172 174346  001363                     bne     4$              ;br if either bit failed to clear
```

```
                                        .sbttl  ..TEST 2 - Program Memory (Vector area)
1175 174350  005237  172340            inc     TESTNO          ;set test # to 2
1176 174354                     MEMTST: ;+
1177                                    ;-------------------------------------------------------------
1178                                    ;
1179                                    ;       Test the first 2048 bytes of program memory
1180                                    ;
1181                                    ;       If a failure occurs in the first 2048 bytes, then switch
1182                                    ;       the memory bank that occupies that address range, if
1183                                    ;       possible, and repeat test.  If the memory bank can not be
1184                                    ;       switched, or if the second bank also fails, declare error.
1185                                    ;
1186                                    ;-------------------------------------------------------------
1187                                    ;-
1188 174354  005004                     clr     R4              ;address 0 is base address of area to test
1189 174356  012705  003776            mov     #VECSIZ-2,R5    ;R5 gets last address to test
1190 174362  005003                     clr     R3              ;R3 gets # of failing addresses to store (none)
1191 174364  004737  176570            jsr     PC,#DUINU       ;call the memory test
1192 174370  103022                     bcc     PARTST          ;br if memory test passed
1193 174372                     SWTTST: ;+
1194                                    ;-------------------------------------------------------------
1195                                    ;
1196                                    ;       if( 'bank swap'=0 )  then ( 'bank swap' gets a 1, retry test )
1197                                    ;       if( 'bank swap'=1 AND 'board swap'=1 ) then ( declare unrecoverable erro
)
1198                                    ;       if( 'bank swap'=1 AND 'board swap'=0 )  then ( 'bank swap' gets a 0,
1199                                    ;                                                      'board swap' gets a 1
1200                                    ;                                                       retry the test      )
1201                                    ;
1202                                    ;NOTE ON PROGRESSION OF 'SWAP BANK' AND 'SWAP BOARD' SIGNALS:
1203                                    ;
1204                                    ;       Pass 1 - 'swap bank' and 'swap board' both zero
1205                                    ;       Pass 2 - 'swap bank' = 1, 'swap board' = 0
1206                                    ;       Pass 3 - 'swap bank' = 0, 'swap board' = 1
1207                                    ;       Pass 4 - 'swap bank' = 1, 'swap board' = 1
1208                                    ;
1209                                    ;-------------------------------------------------------------
1210                                    ;-
1211 174372  012706  000006            mov     #SAVRS5,SP      ;reset SP
1212 174376  012703  000006            mov     #SP1CS,R3       ;R3 points to P.iec Control and Status register
1213 174402  032713  002000            bit     #SWPBNK,(R3)    ;are banks swapped now?
1214 174406  001003                     bne     10$             ;br if banks are now swapped
1215 174410  052713  002000            bis     #SWPBNK,(R3)    ;assert 'swap banks' signal
1216 174414  000757                     br      MEMTST          ;retry memory test on first 2048 bytes
1217
1218 174416  032713  004000        10$:  bit     #SWPBRD,(R3)    ;are boards swapped now?
1219 174422  001061                     bne     BADMEM          ;branch if banks and boards are both swapped already
1220 174424  042713  002000            bic     #SWPBNK,(R3)    ; 'swap banks' gets a 0
1221 174430  052713  004000            bis     #SWPBRD,(R3)    ; 'swap boards' gets a 1
1222 174434  000747                     br      MEMTST          ;retry memory test
```

```
                                        .sbttl  ..TEST 3 - Program Memory (Partition area)
174436                          PARTST: ;+
                                        ;-------------------------------------------------------------
                                        ;
                                        ;       Find a working 8 kword partition in Program Memory
                                        ;
                                        ;       If no failures are detected, then continue with TU58 test.
                                        ;       If a failure is detected, set the base address of the area
                                        ;       to be tested to the next address following the failure, and
                                        ;       try to find a contiguous 8 kword area with no errors.  If no
                                        ;       8 kword area can be found, declare a memory test error.
                                        ;
                                        ;-------------------------------------------------------------
                                        ;-
174436  005237  172340                  inc     TESTNO          ;set test # to 3
174442  012706  004000                  mov     #VECSIZ,SP      ;set stack pointer to top of tested 2048 bytes
174446  012737  174514  000004          mov     #MEMERR,4       ;set timeout trap vector
174454  012737  000340  000006          mov     #340,6          ;set trap 4 psw word
174462  012737  174514  000114          mov     #MEMERR,#0114   ;set parity trap vector
174470  012737  000340  000116          mov     #340,#0116
174476  012704  004000                  mov     #VECSIZ,R4      ;R4 gets first address to test
174502  012705  043776                  mov     #VECSIZ+PARSIZ-2,R5 ;R5 gets last address to test
174506  004737  176570          MEMRTY: jsr     PC,#MEMINV      ;perform the Moving-Inversions test
174512  103023                          bcc     MEMOKM          ;br if memory test passed
                                        ;+
                                        ;-------------------------------------------------------------
                                        ;
                                        ;       R0 contains the memory address that failed.
                                        ;
                                        ;       Add 2 to R0, then adjust R0 upwards (if necessary) to the
                                        ;       next even 32 word boundary.  Start the test over there.
                                        ;
                                        ;       if (last address of next partition is beyond the end of memory)
                                        ;       then ( declare a program memory failure)
                                        ;       else ( test the newly defined partition)
                                        ;
                                        ;-------------------------------------------------------------
                                        ;-
174514  012706  004000          MEMERR: mov     #VECSIZ,SP      ;reset SP in case we trapped to here
174520  010004                          mov     R0,R4           ;move failing address to R4
174522  062704  000002                  add     #2,R4           ;adjust address past failure
174526  032704  000077                  bit     #77,R4          ;is address on 32 word boundary?
174532  001404                          beq     10$             ;br if it is
174534  042704  000077                  bic     #77,R4          ;adjust address to 32 word boundary
174540  062704  000100                  add     #100,R4
174544  010405                  10$:    mov     R4,R5           ;set R5 to starting address
174546  062705  037776                  add     #PARSIZ-2,R5    ;R5 now = last address of 8 kword partition to test
174552  020527  160000                  cmp     R5,#160000      ;see if we are into I/O page yet
174556  103003                          bhis    #BADMEM         ;br if we are into I/O page
174560  000752                          br      MEMRTY          ;retry memory test on new partition
                                        ;
174562  010400                  MEMOKM: mov     R4,R0           ;R4 gets base address of 8 kword partition
174564  000437                          br      TSTST           ;now go do TU58 test and load
```

```
                                        .sbttl  FAULT Lamp Routine
                                        .enabl  lsb
174566                          BADMEM: ;+
                                        ;-------------------------------------------------------------
                                        ;
                                        ;       Declare a memory test error.
                                        ;       Set the fault code register (R0) to 'Memory Module Failure'
                                        ;
                                        ;-------------------------------------------------------------
                                        ;-
174566  012700  000022                  mov     #MEMFAL,R0      ;R0 gets code for 'memory module failure'
174572                          ANYFAL: ;+
                                        ;-------------------------------------------------------------
                                        ;
                                        ;       Handle Fault lamp report for any error. R0 contains failure code
                                        ;
                                        ;       Clear all lamps except INIT (indicates whether F-11 tests passed)
                                        ;       Light the Fault lamp, then monitor the fault switch. When the Fault
                                        ;       switch is pressed, display a blinking failure code in the lamps.
                                        ;
                                        ;-------------------------------------------------------------
                                        ;-
174572  106427  000340                  mtps    #340            ;disable interrupts
174576  012701  0000006                 mov     #BDBCSR,R1      ;point R1 to P.ioc display control register
174602  042711  000017                  bic     #FLTLMP!BLKLMP!HBLMP!HALMP,(R1) ;extinguish all lamps except INIT
174606  052711  000010                  bis     #FLTLMP,(R1)    ;light the fault lamp
174612  032711  004000          400:    bit     #FLTSW,(R1)     ;has the fault switch been depressed?
174616  001775                          beq     400             ;br if not
                                        ;+
                                        ;-------------------------------------------------------------
                                        ;
                                        ;Display the fault code in the lamps for 1/2 second
                                        ;
                                        ;-------------------------------------------------------------
                                        ;-
174620  042711  000037          500:    bic     #ALLLMP,(R1)    ;extinguish all lamps
174624  010011                          mov     R0,(R1)         ;put fault code in lamps
174626  012703  000012          PTIME0: mov     #10.,R3         ;repeat 50 msec delay below 10 times
174632  012702  045570          600:    mov     #19320.,R2      ;R2 gets constant for 50 msec delay
174636  077201                  700:    sob     R2,700          ;delay 50 msec
174640  077304                          sob     R3,600          ;delay 1/2 second (500 msec) total
                                        ;+
                                        ;-------------------------------------------------------------
                                        ;
                                        ;Extinguish all the lamps for 1/2 second
                                        ;
                                        ;-------------------------------------------------------------
                                        ;-
174642  042711  000037                  bic     #ALLLMP,(R1)    ;turn off all lamps
174646  012703  000012          PTIME1: mov     #10.,R3         ;repeat 50 msec delay below 10 times
174652  012702  045570          800:    mov     #19320.,R2      ;R2 gets constant for 50 msec delay
174656  077201                  900:    sob     R2,900          ;delay 50 msec
174660  077304                          sob     R3,800          ;delay 1/2 sec (500 msec) total
174662  000756                          br      500             ;turn on the lamps for 1/2 second again
                                        .dsabl  lsb
```

```
                        .sbttl  ..TEST 4 - TU58 Tests
                ;+
                ;*********************************************************************
                ;       Find a working TU58 tape drive with media mounted and
                ;       read the first 0 blocks to the BR partition just tested.
                ;
                ;Inputs:  R0 = Base address of the 0 Kword partition(supplied by memory test)
                ;         MMU Kernel PAR 5 = Unit # of TU58(lower byte) and type of boot (upper bit)
                ;         MMU Kernel PAR 6 = CSR address of TU58 to use for boot (16 bits)
                ;         MMU Kernel PAR 7 = Validation word for unit and CSR
                ;
                ;Process:  Find a working TU58 tape drive that can be used to read in the Init P.ioc test.
                ;       First, a check is made to see if we were passed a specific TU58 CSR and unit to us
e.
                ;       The parameters in MMU Kernel PAR's 5, 6 and 7 are checked for validity by testing
if
                ;       the validation word is equal to the XOR of the Unit & CSR words  If the parameters
                ;       are valid, the load is attempted via the TU58 specified.  If the parameters in the
                ;       PAR's are not valid, or if the load attempt fails on the tape specified, the defau
lt
                ;       selection sequence (described below) is used.
                ;
                ;       DEFAULT SELECTION SEQUENCE
                ;       Any of the 4 TU58 drives that can be connected to the P.ioc can be used as a
                ;       boot device. Testing begins with TU58 controller 01, unit 0.  If unit 0 does not
                ;       respond, unit 1 is tried.  If unit 1 also fails, the second TU58 controller is
                ;       tested; unit 0 first, then unit 1 if unit 0 fails.  Finally, the serial inter-
                ;       face to the local terminal is tried if all 4 TU58 drives are unusable for booting.
                ;       Both unit 0 and unit 1 are used in this case also.
                ;
                ;       The test consists of directing the selected TU58 unit to perform its
                ;       self test.  If the self-test fails, the next unit in sequence is tried.
                ;       If the self-test passes, the unit is directed to read its first 0
                ;       blocks to the 0 Kword partition in Program memory.  If the first word
                ;       of the loaded image is a NOP instruction, control is passed to the
                ;       first instruction of the loaded image.  If the first instruction is not
                ;       a NOP, then the next TU58 unit in sequence is tried.
                ;
                ;       Failure to find any working TU58 drive with media mounted is a fatal
                ;       error, and results in the failure action listed below (see  outputs ).
                ;
                ;Outputs:  Case 01 - A working TU58 with bootable media mounted was found:
                ;               Parameters passed:
                ;               a) Type of boot (warm or cold)
                ;               b) Base address of tested 0 Kword partition
                ;               c) Unit number of the TU58 that was used to load
                ;               d) CSR address of the TU58 that was used to load
                ;
                ;          Case 02 - No working TU58 could be found:
                ;
                ;               State of F-11:
                ;               a) 'Init' lamp is lit
                ;               b) 'Fault' lamp is lit
                ;               c) F-11 in loop monitoring the 'fault lamp' . Depressing
                ;                  the 'fault lamp' results in a blinking display of the
                ;                  fault code for 'TU58 Failure'.
                ;               d) F-11 must be rebooted to attempt recovery
                ;
                ;               Probable failure:
                ;               a) No MSCP system cassette is mounted
```

```
1392                            ;               b) Faulty TU58 unit (if boot fails on just one unit)
1393                            ;               c) Faulty TU58 controller (if boot fails on both units)
1394                            ;
1395                            ;               Recovery action:
1396                            ;               a) Mount an HSC50 system cassette (if none was mounted) and reboot.
1397                            ;               b) Switch the HSC50 system cassette to the other TU58 unit.
1398                            ;                  (if proper media was mounted)
1399                            ;               c) Refer to description of TU58 error table in the USEFUL INFORMATION
1400                            ;                  section of this listing.
1401                            ;
1402                            ;
1403                            ;Notes:
1404                            ;
1405                            ;History of changes to this test:
1406                            ;
1407                            ;*********************************************************************************
1408                            ;-
1409
```

```
                                        .enabl  lsb
1412 174664  005237  172340    t58tst: inc     TESTNO          ;set test # to 4
                                        ;+
                                        ;-------------------------------------------------------------
                                        ;
                                        ;       Set the stack pointer (R6) to the highest location
                                        ;       tested.  Note that since we only load 8(10) blocks,
                                        ;       we avoid overwriting this area when reading from tape.
                                        ;
                                        ;-------------------------------------------------------------
                                        ;-
1422 174670  010006                     mov     R0,SP           ;point stack to lowest address of 8K partition
1423 174672  062706  040000             add     #PARSIZ,SP      ;adjust to last address (+2) of 8K partition
                                        ;+
                                        ;-------------------------------  -------------------------------
                                        ;
                                        ;       See if a TU58 unit and CSR was specified for booting.
                                        ;
                                        ;       if (unit and CSR are valid) then ( try specified unit)
                                        ;       else ( use default selection sequence)
                                        ;
                                        ;       if (specified unit fails) then (use default sequence)
                                        ;
                                        ;-------------------------------------------------------------
                                        ;-
1436 174676  013704  000006             mov     @#KPAR5,R4      ;R4 gets the unit word (upper bit=type of boot)
1437 174702  042704  100000             bic     #100000,R4      ;clear out type of boot field
1438 174706  013702  000006             mov     @#KPAR6,R2      ;R2 gets the CSR word
1439 174712  074402                     xor     R4,R2           ;R2 gets XOR of unit and CSR words
1440 174714  020237  000006             cmp     R2,@#KPAR7      ;is specified unit and CSR valid?
1441 174720  001020                     bne     50              ;br if not and use default sequence
1442 174722  042704  177400             bic     #177400,R4      ;clear out upper byte of unit word
1443 174726  013701  000006             mov     @#KPAR6,R1      ;R1 gets CSR address for TU58 subroutine use
1444 174732  020127  160000             cmp     R1,#160000      ;insure CSR address in I/O page
1445 174736  103411                     blo     50              ;br if CSR is not in I/O page, use default sequence
1446 174740  020427  000001             cmp     R4,#1           ;insure unit # is not greater than 1
1447 174744  003006                     bgt     50              ;br if unit # is greater than 1, use default seq
1448 174746  004737  175226             jsr     PC,T58SST       ;Perform sync and self-test on TU58
1449 174752  103403                     bcs     50              ;br if sync or test failed, use default sequence
1450 174754  004737  175362             jsr     PC,READBT       ;read boot blocks and check for bootable image
1451 174760  103054                     bcc     trnsfr          ;br if read ok, and bootable image loaded
1452                                    ;(else we fall through to default sequence below)
```

```
1454
1455 174762  012746  175074          50:     mov     @CSRLST,-(SP)   ;stack a pointer to the CSR list
1456 174766  012737  000374  000422          mov     @SLIERR-4,TUERPT ;point to TU 01's error table entry (-4)
1457                                         ;+
1458                                         ;-------------------------------------------------------------
1459                                         ;
1460                                         ;       Select next TU58 controller for bootstrap. The controllers are
1461                                         ;       tried in the following order:
1462                                         ;               1) TU58 01
1463                                         ;               2) TU58 02
1464                                         ;               3) Console interface
1465                                         ;
1466                                         ;       If ( all controllers have been tried )
1467                                         ;       Then ( declare a TU58   ilure )
1468                                         ;
1469                                         ;-------------------------------------------------------------
1470                                         ;-
1471 174774  005037  000420          100:    clr     TUERR0          ;clear TU58 SW/HW error codes
1472 175000  062737  000004  000422          add     #4,TUERPT       ;update pointer to next error table entry
1473 175006  017601  000000                  mov     @0(SP),R1       ;R1 gets RCSR address of next controller to use
1474 175012  001434                          beq     T58FAL          ;br if all controllers have been tried
1475 175014  062716  000002                  add     #2,(SP)         ;update pointer to next CSR for next time around
1476                                         ;+
1477                                         ;-------------------------------------------------------------
1478                                         ;
1479                                         ;       Synchronize with the TU58 and issue self-test command
1480                                         ;
1481                                         ;       Set R4 = 0 to indicate unit 0 (not required for sync, but used
1482                                         ;                               in the 'READBT' routine called below.)
1483                                         ;
1484                                         ;-------------------------------------------------------------
1485                                         ;-
1486 175020  005004                          clr     R4              ;set unit 0 to zero
1487 175022  004737  175226                  jsr     PC,T58SST       ;call the sync and self-test routine
1488 175026  103003                          bcc     150             ;br if sync and self-test passed
1489 175030  004737  176544                  jsr     PC,SAVERR       ;save error codes in table
1490 175034  000757                          br      100             ;try the next controller
1491
1492 175036                          150:    ;+
1493                                         ;-------------------------------------------------------------
1494                                         ;
1495                                         ;       First try to read a bootable image from unit 0 of the
1496                                         ;       currently selected controller.  If we can't load a bootable
1497                                         ;       image from unit 0, try unit 1.
1498                                         ;
1499                                         ;       (If the read succeeds, we have read the boot blocks and the
1500                                         ;        TU58 directory into the 8 kword partition.)
1501                                         ;
1502                                         ;-------------------------------------------------------------
1503                                         ;-
1504 175036  005037  000420          200:    clr     TUERR0          ;clear TU58 HW/SW error codes
1505 175042  004737  175362                  jsr     PC,READBT       ;read boot blocks and check for bootable image
1506 175046  103010                          bcc     300             ;br if read successful and image is bootable
1507 175050  004737  176544                  jsr     PC,SAVERR       ;save error codes in table
```

```
1509                                    ;+
1510                                    ;-------------------------------------------------------------
1511                                    ;
1512                                    ;       If ( Unit = 1 ) Then ( Try the next controller )
1513                                    ;       Else ( set Unit to 1, try booting from unit 1 )
1514                                    ;
1515                                    ;-------------------------------------------------------------
1516                                    ;-
1517 175054  005704                     tst     R4              ;is unit 0 zero now ?
1518 175056  001346                     bne     100             ;br if not and try next controller
1519 175060  005204                     inc     R4              ;set unit to 1
1520 175062  004737  175630             jsr     PC,T58SYC       ;re-sync with controller
1521 175066  000763                     br      200             ;try reading from unit 1
1522
1523                                    ;+
1524                                    ;-------------------------------------------------------------
1525                                    ;
1526                                    ;       Remove pointer from stack and continue boot
1527                                    ;
1528                                    ;-------------------------------------------------------------
1529                                    ;-
1530 175070  005726             300:    tst     (SP)+           ;pop pointer from stack
1531 175072  000407                     br      T58OFR
1532
1533
1534                                    ;list of TU58 CSR's for booting
1535 175074  0000006             CSRLST: .word   SSL1RS  ;TU58 #1 RCSR address
1536 175076  0000006                     .word   SSL2RS  ;TU58 #2 RCSR address
1537 175100  0000006                     .word   SSL0RS  ;console RCSR address
1538 175102  000000                      .word   0       ;list terminator
1539
1540
1541
1542 175104                      T58FAL: ;+
1543                                    ;-------------------------------------------------------------
1544                                    ;
1545                                    ;       Declare TU58 error.
1546                                    ;
1547                                    ;       1)  Set fault code to 'TU58 failure'
1548                                    ;
1549                                    ;       2)  Light the fault lamp
1550                                    ;
1551                                    ;       3)  Enter loop to monitor fault lamp
1552                                    ;
1553                                    ;-------------------------------------------------------------
1554                                    ;-
1555 175104  012700  000023             mov     #TU58FL,R0      ;R0 gets error code for TU58 failure
1556 175110  000630                     br      ANYFAL          ;go to common error reporter
1557                                    .dsabl  lsb
```

```
                        .sbttl  ..TEST 5 - Transfer Control to Loaded Image

                ;+
                ;*****************************************************************************
                ;
                ;       The image just loaded to Program Memory is either the 'Init P.IOC' test,
                ;       or an non-HSC50 bootstrap (e.g. RT-11, RSX-11).  In either case, the image
                ;       must be 'massaged' to work properly.  In the case of the 'Init P.IOC' test,
                ;       a one block (512 byte) 'hole' exists between the first and second blocks
                ;       of the test, due to the presence of a 'Volume ID' block on the TU58.  This
                ;       'hole' is collapsed before transferring control to the loaded image.  In the
                ;       case of a 'foreign' (non-HSC50) bootstrap, the image must be moved from
                ;       the tested 8 Kword partition down to address 000000 to work properly.
                ;
                ;       After 'massaging' the loaded image, set up the parameters to be passed and
                ;       transfer to the image.  For the 'Init P.IOC' test, transfer to the base address
                ;       of the loaded image.  For a 'foreign' bootstrap, transfer control to location 0
                ;
                ;Inputs:        All tests passed (implied)
                ;               TU58 parameters (CSR, Unit 0)
                ;               Base address of working 8 Kword partition
                ;               Type of boot (warm or cold)
                ;               'Init' lamp is lit
                ;               'Fault' lamp is off
                ;
                ;Process:       Check the second instruction of the image just loaded.
                ;
                ;               if ( second instruction = NOP ) then ( Init P.IOC test was loaded )
                ;               else ( a foreign bootstrap was loaded )
                ;
                ;               if ( Init P.IOC test was loaded)
                ;               then ( collapse the 1 block hole between first and second blocks,
                ;                      transfer to first address loaded)
                ;
                ;               if ( foreign boot was loaded )
                ;               then ( move image down to location 0, transfer to loc 0 )
                ;
                ;Outputs:       See 'Inputs' above
                ;
                ;Notes:
                ;
                ;History of changes to this routine:
                ;
                ;*****************************************************************************
                ;-
```

```
                                        .enabl  lsb
175112                          TRNSFR: ;+
                                        ;-------------------------------------------------------------
                                        ;
                                        ;       Set up the parameters that specify the TU58 unit and CSR used for
                                        ;       the load just completed.  They may have changed if the unit
                                        ;       specified for booting was not working.
                                        ;
                                        ;-------------------------------------------------------------
                                        ;-
175112  005237  172340                  inc     TESTNO          ;set test # to 5
175116  042737  077777  000006          bic     #77777,@#KPAR5  ;mask everything except type of boot bit
175124  110437  000006                  movb    R4,@#KPAR5      ;save unit # we booted from
175130  010137  000006                  mov     R1,@#KPAR6      ;save CSR of TU58 we booted from
                                        ;+
                                        ;-------------------------------------------------------------
                                        ;
                                        ;       Determine if we loaded the 'Init P.IOC' test or a foreign bootstrap
                                        ;       if (second instruction of loaded image = NOP)
                                        ;       then ( Init P.IOC test was loaded)
                                        ;       else ( Foreign boot was loaded )
                                        ;
                                        ;-------------------------------------------------------------
                                        ;-
175134  010002                          mov     R0,R2           ;R2 gets base address of 8 Kword partition
175136  026227  000002  000240  10$:    cmp     2(R2),#000240   ;is second instruction a NOP ?
175144  001017                          bne     50$             ;br if not, must be a foreign bootstrap
                                        ;+
                                        ;-------------------------------------------------------------
                                        ;
                                        ;       We have loaded the 'Init P.IOC' test.
                                        ;       The 8 blocks loaded are as follows:
                                        ;
                                        ;       block 0         first 512 bytes of 'Init P.ioc' test
                                        ;       block 1         RT-11 volume ID block
                                        ;       block 2         second 512 bytes of 'Init P.ioc' test
                                        ;       block 3         third 512 bytes of 'Init P.ioc' test
                                        ;       block 4         fourth 512 bytes of 'Init P.ioc' test
                                        ;       block 5         fifth 512 bytes of 'Init P.ioc' test
                                        ;       block 6         first half of first RT-11 directory segment
                                        ;       block 7         second half of first RT-11 directory segment
                                        ;       We must move blocks 2 thru 7 down one block to collapse the hole.
                                        ;-------------------------------------------------------------
                                        ;-
175146  010203                          mov     R2,R3           ;R3 gets base address used for load
175150  062703  001000                  add     #1000,R3        ;R3 now points to 'hole'
175154  010304                          mov     R3,R4           ;R4 gets R3
175156  062704  001000                  add     #1000,R4        ;R4 now points to first address beyond 'hole
175162  012705  003000                  mov     #LOOSIZ-2*400,R5 ;R5 will count words of data to be moved
175166  012423                  20$:    mov     (R4)+,(R3)+     ;move a word
175170  077502                          sob     R5,20$          ;branch until blocks are moved
```

```
                                ;+
                                ;-------------------------------------------------------------
                                ;
                                ;       Set up parameters to be passed to the 'INIT P.IOC' test.
                                ;
                                ;       R0 = Unit # of TU58 used to boot, and 'type of boot'
                                ;       R1 = CSR address of TU58 used to boot
                                ;       R2 = Base address of the tested 8 kword partition
                                ;
                                ;-------------------------------------------------------------
                                ;-
1667 175172  013700  000006     mov     %0%R5,R0        ;R0 gets unit # and type of boot
1668 175176  013701  000006     mov     %0%R6,R1        ;R1 gets CSR address of TU58 used for boot
1669 175202  000112             jmp     (R2)            ;transfer to the Init P.ioc test
1670
                                ;+
                                ;-------------------------------------------------------------
                                ;
                                ;       The image just loaded is a foreign (non-HSC50) bootstrap.  The image was
                                ;       loaded to the 8 kword partition in Program memory, but is designed to
                                ;       execute from locations 000000 thru 000776. So we move the first block
                                ;       from the 8 kword partition, down to address 000000, before jumping
                                ;       to address 000000 to begin the bootstrap.
                                ;
                                ;-------------------------------------------------------------
                                ;-
1682 175204  010203     500:    mov     R2,R3           ;R3 gets base address of load
1683 175206  005004             clr     R4              ;R4 points to where to start moving data
1684 175210  012705  000400     mov     #400,R5         ;set R5 to count 1 block of data
1685 175214  012324     600:    mov     (R3)+,(R4)+     ;move a word
1686 175216  077502             sob     R5,600          ;branch until block is moved
1687                             ;+
                                ;-------------------------------------------------------------
                                ;
                                ;       Now jump to location 0 to continue foreign bootstrap
                                ;
                                ;-------------------------------------------------------------
                                ;-
1694 175220  113700  000006     movb    %0%R6,R0        ;R0 gets unit # word
1695 175224  005007             clr     PC              ;jump to 0 to start foreign bootstrap
1696                             .dsabl  lsb
```

```
                        .sbttl  Issue TU58 Sync and Self-Test

                ;+
                ;*********************************************************************************
                ;
                ;       Issue TU58 synchronize and self-test sequence
                ;
                ;inputs:
                ;       R0 = Base address of tested partition in Program Memory
                ;       R1 = CSR address of TU58 to use
                ;       R2,R3,R5 are SCRATCH
                ;
                ;process:
                ;       1) Perform time-out test on TU58 I/O registers.
                ;       2) Issue TU58 synchronization sequence.
                ;       3) If (sync fails) then (exit with c bit set)
                ;       4) Issue TU58 self-test command
                ;       5) If (self-test fails) Then (exit with c bit set)
                ;          Else (exit with c bit clear)
                ;       NOTE: Loop-back mode can not be used as a test, due to lack of space
                ;
                ;outputs:
                ;       R0,R1 are PRESERVED
                ;       R2,R3, and R5 are CLOBBERED
                ;
                ;*********************************************************************************
                ;-

                        .enabl  lsb
175226          T58SST: ;+
                        ;---------------------------------------------------------------
                        ;
                        ;       Check for Non-Existent-Memory trap when accessing the four
                        ;       TU58 I/O registers.  If any one of the four addresses results
                        ;       in a NXM trap, exit this routine with C bit set.
                        ;
                        ;---------------------------------------------------------------
                        ;-
175226  013746  000004          mov     @#4,-(SP)       ;save the present contents of the NXM vector
175232  012737  175344  000004  mov     #300$,@#4       ;replace NXM vector with address of 300 below
175240  010102                  mov     R1,R2           ;R2 gets address of TU58 receiver status register
175242  005722                  tst     (R2)+           ;test recvr status reg
175244  005722                  tst     (R2)+           ;test recvr buffer
175246  005722                  tst     (R2)+           ;test xmitter status
175250  005722                  tst     (R2)+           ;test xmitter buffer
175252  012637  000004  100$:   mov     (SP)+,@#4       ;restore NXM vector
175256  103431                  bcs     20$             ;br if we had a timeout
                        ;+
                        ;---------------------------------------------------------------
                        ;
                        ;       Perform TU58 synchronization sequence
                        ;
                        ;---------------------------------------------------------------
                        ;-
175260  004737  175630          jsr     PC,T58SYC       ;sync with TU58
175264  103426                  bcs     20$             ;br if sync failed
                        .*
```

```
                                ;-----------------------------------------------------------------
                                ;
                                ;       Issue self test command to TU58 unit
                                ;       The F-11 sends a 'Diagnose' command to the TU58 unit and awaits
                                ;       an end-packet from the TU58.  If the end-packet arrives with an
                                ;       error indicated, or if no response is received within 40
                                ;       seconds, exit this routine with the C bit set.  If the end-
                                ;       packet arrives with success indicated, exit with c bit clear.
                                ;
                                ;-----------------------------------------------------------------
                                ;-
1766 175266  012703  176704    mov     @digpak,R3      ;point R3 to a 'Diagnose' command packet
1767 175272  004737  176136    jsr     PC,sendpk       ;send 'diagnose' command packet
1768 175276  103421            bcs     20$             ;br if transmission failed
                                ;+
                                ;-----------------------------------------------------------------
                                ;
                                ;       Get result of self test, by receiving an 'end' packet
                                ;
                                ;       if (test fails) then (set c bit, exit)
                                ;
                                ;-----------------------------------------------------------------
                                ;-
1778 175300  010003            mov     R0,R3           ;construct pointer to packet buffer in R3
1779 175302  062703  037400    add     @#RSIZ-256.,R3  ;R3 now points to a safe area for packet buffer
1780 175306  010305            mov     R3,R5           ;save pointer to packet buffer in R5
1781 175310  004737  176242    jsr     PC,recvpk       ;receive a packet at (R3)
1782 175314  103412            bcs     20$             ;br if reception failed
1783 175316  105765  000003    tstb    3(R5)           ;test for successful self-test(positive if success)
1784                            ;clc
1785 175322  100007            bpl     20$             ;br if self-test passed, exit with c clear
1786 175324  116537  000003 000421  movb    3(R5),TUSER     ;save error code from end packet
1787 175332  112737  000003 000420  movb    @TUSLFT,TUSER   ;save software error code (self-test failed)
1788 175340  000261            10$:    sec             ;set c bit to indicate failure
1789 175342  000207            20$:    rts     PC              ;exit
1790
1791                            ;+
1792                            ;-----------------------------------------------------------------
1793                            ;
1794                            ;       Enter here if we get a NXM trap while accessing the
1795                            ;       TU58 I/O page registers.  Clean up stack and set c bit.
1796                            ;
1797                            ;-----------------------------------------------------------------
1798                            ;-
1799 175344  062706  000004    30$:    add     #4,SP           ;remove PC and PSW from stack
1800 175350  112737  000001 000420  movb    @TUNXM,TUSER    ;save software error code (NXM trap accessing CSR's)
1801 175356  000261            sec                     ;set c bit to indicate error
1802 175360  000734            br      10$
1803                            .dsabl  lsb
```

```
                        .sbttl  Read TU58 Boot Blocks to Program Memory

                ;+
                ;**********************************************************************
                ;
                ;       Read TU58 boot blocks to Program Memory and test for Bootable image
                ;
                ;inputs:
                ;       R0 = Base address of tested Program Memory partition
                ;       R1 = CSR address of TU58 to be used
                ;       R2 = SCRATCH
                ;       R3 = SCRATCH
                ;       R4 = Unit # of TU58 to be used
                ;       R5 = SCRATCH
                ;
                ;process:
                ;       Issue a TU58 read command for 8  blocks, load the data beginning at
                ;       the address in R0.
                ;       NOTE: Of the 8 blocks read, 5 are boot blocks, 1 is volume ID block,
                ;             2 are directory blocks.
                ;
                ;outputs:
                ;       C clear if read is successful (no errors and first instruction = NOP)
                ;       C set if read fails or first instruction is not a NOP
                ;       R0,R1, and R4 are PRESERVED
                ;       R2,R3, and R5 are CLOBBERED
                ;
                ;**********************************************************************
                ;-
READBT: ;+
                ;----------------------------------------------------------------------
                ;
                ;       call the TU58 loader
                ;
                ;----------------------------------------------------------------------
                ;-
1041 175362 010046              mov     R0,-(SP)        ;save base address of partition on stack
1042 175364 010005              mov     R0,R5           ;construct packet buffer ptr in R5
1043 175366 062705 037400       add     #PKTSIZ-256.,R5
1044 175372 010002              mov     R0,R2           ;R2 gets starting address for load
1045 175374 010400              mov     R4,R0           ;R0 gets unit # of TU58
1046 175376 010503              mov     R5,R3           ;R3 gets pointer to packet buffer
1047 175400 012704 000010       mov     #LOGSIZ,R4      ;R4 gets # of blocks to load
1048 175404 005005              clr     R5              ;R5 gets block # of 0
1049 175406 004737 175442       jsr     PC,TSDREAD      ;call the read subroutine
1050 175412 010004              mov     R0,R4           ;restore unit # to R4
1051 175414 012600              mov     (SP)+,R0        ;restore base address in R0
1052 175416 103410              bcs     30$             ;br if load failed (device error)
```

```
                                        ;+
                                        ;-------------------------------------------------------------------
                                        ;
                                        ;       Check for a bootable image just loaded.  A bootable image
                                        ;       contains a PDP-11 NOP instruction in the first word loaded.
                                        ;
                                        ;-------------------------------------------------------------------
                                        ;-
1862 175420  021027  000240            cmp     (R0),#000240    ;is first instruction of image loaded a NOP ?
1863 175424  001404                    beq     20$             ;br if yes, we have a bootable image
1864 175426  112737  000004  000420    movb    #TUSUFT,TUBUFR  ;remember that we did not find a bootable image
1865 175434  022707                10$: cmp     (PC)+,PC        ;set c bit, skip next instruction
1866 175436  000241                20$: clc                     ;clear c bit
1867 175440  000207                30$: rts     PC              ;return
```

```
                        .sbttl  TU58 Read Routine
                ;+
                ;**********************************************************************
                ;
                ;       Read from a TU58 to Program Memory
                ;
                ;inputs:
                ;       R0 = TU58 Unit 0
                ;       R1 = TU58 RCSR address
                ;       R2 = Starting address for load
                ;       R3 = Address of a packet buffer
                ;       R4 = # of blocks to read (load)
                ;       R5 = Starting block for read
                ;
                ;process:
                ;       Construct a read command packet and send it to the TU58.  Load the
                ;       data read from the tape to Program Memory, and check for a successful
                ;       read.  condition c bit accordingly.
                ;
                ;outputs:
                ;       R0,R1,R2, and R3 are PRESERVED
                ;       R3,R4, and R5 are CLOBBERED
                ;       C bit is clear is read and image check both succeed.
                ;       C bit set if read fails, or if image is not bootable.
                ;
                ;**********************************************************************
                ;-
175442          TSBREAD: ;+
                        ;--------------------------------------------------------------
                        ;
                        ;       Preserve R0,R1,R2, and R3 on the stack
                        ;
                        ;--------------------------------------------------------------
                        ;-
175442  010046          mov     R0,-(SP)        ;preserve unit 0
175444  010246          mov     R2,-(SP)        ;preserve load start address
175446  010346          mov     R3,-(SP)        ;preserve packet buffer address
                        ;+
                        ;--------------------------------------------------------------
                        ;
                        ;       Construct a read command in the packet buffer
                        ;
                        ;--------------------------------------------------------------
                        ;-
175450  112723  000002  movb    #CMD,(R3)+      ;flag byte = 'command'
175454  112723  000012  movb    #10.,(R3)+      ;byte count = 10
175460  112723  000002  movb    #READ,(R3)+     ;op-code = 'read'
175464  105023          clrb    (R3)+           ;modifier = 0
175466  110023          movb    R0,(R3)+        ;insert unit 0
175470  105023          clrb    (R3)+           ;switches = 0
175472  105023          clrb    (R3)+           ;seq # (low) = 0
175474  105023          clrb    (R3)+           ;seq # (high) = 0
175476  012700  000011  mov     #9.,R0          ;use R0 to count 9 shifts
175502  006304          asl     R4              ;shift block count 9 times to multiply by 1000(8)
175504  077002          sob     R0,10$          ;(block count times 1000 = byte count)
175506  005704          tst     R4              ;insure a non-zero number of bytes to load
175510  001442          beq     40$             ;take error exit if byte count is zero
```

```
                                        movb    R4,(R3)+        ;save byte count (low)
                                        swab    R4              ;reverse bytes in R4
                                        movb    R4,(R3)+        ;save byte count (high)
                                        movb    R5,(R3)+        ;save block # (low)
                                        swab    R5              ;reverse bytes in R5
                                        movb    R5,(R3)+        ;save block # (high)
                                        ;+
                                        ;-------------------------------------------------------------------------
                                        ;
                                        ;       Send the read command packet to the TU58
                                        ;
                                        ;-------------------------------------------------------------------------
                                        ;-
                                        mov     (SP),R3         ;point R3 to packet just constructed
                                        jsr     PC,SENDPK       ;call the send packet routine
                                        bcs     40$             ;br if send packet failed
                                        ;+
                                        ;-------------------------------------------------------------------------
                                        ;
                                        ;       Accept data packets from TU58
                                        ;
                                        ;-------------------------------------------------------------------------
                                        ;-
                                        mov     2(SP),R2        ;R2 gets starting address for load
                                20$:    mov     (SP),R3         ;point R3 to a packet buffer
                                        jsr     PC,RECVPK       ;call receive packet to input data and command messages
                                        bcs     40$             ;br if reception failed
                                        ;+
                                        ;-------------------------------------------------------------------------
                                        ;
                                        ;       If (last packet = data packet) then (receive another packet)
                                        ;       If (last packet = end packet) then (check for success code)
                                        ;       If (last packet not data or end packet) then (error)
                                        ;
                                        ;-------------------------------------------------------------------------
                                        ;-
                                        mov     (R3),R3         ;point R3 to buffer where last packet was inputted
                                        cmpb    (R3),#DATA      ;check for data packet
                                        beq     20$             ;br if data packet, go get another packet until 'end'
                                        cmpb    2(R3),#ENDP     ;check for end packet
                                        beq     25$             ;br if end packet
                                        movb    #TUPROT,TUBYER  ;remember we got an unknown packet type from the TU
                                        br      40$
                                25$:    tstb    3(R3)           ;check for success code in end packet
                                        ;clc
                                        bpl     30$             ;br if end packet indicates success
                                        movb    3(R3),TUBYER    ;save error code from end packet
                                40$:    sec
                                30$:    mov     (SP)+,R3        ;restore packet buffer pointer
                                        mov     (SP)+,R2        ;restore load starting address
                                        mov     (SP)+,R0        ;restore unit #
                                        rts     PC              ;return
```

```
                        .sbttl  Synchronize with a TU58
                ;+
                ;*****************************************************************************
                ;
                ;       Subroutine to synchronize with a TU58
                ;
                ;inputs.
                ;       R1 = RCSR address of TU58 to synchronize
                ;       R2 = SCRATCH
                ;       R3 = SCRATCH
                ;process:
                ;       Send a 'break' to the TU58 for 1 character time  Then send 2  Init' characters
                ;       Wait up to 2 seconds for the TU58 to respond with a  Continue
                ;outputs:
                ;       R1 = PRESERVED
                ;       R2 = CLOBBERED
                ;       R3 = CLOBBERED
                ;
                ;*****************************************************************************
                ;-
                        .enabl  lsb
175630          T58SYC: ;+
                        ;-------------------------------------------------------------
                        ;
                        ;       Synchronize the F-11 with the TU58.  The F-11 sends a 'break'
                        ;       character to the TU58 for one character time, then the F-11
                        ;       sends two TU58 'Init' commands and waits 40 seconds for the
                        ;       TU58 to respond with a 'Continue' message.  If the F-11 does
                        ;       not respond correctly, exit this routine with the c bit set,
                        ;       otherwise exit with c bit clear
                        ;
                        ;-------------------------------------------------------------
                        ;-
175630  012761  000001  000004        mov     #tubrk,4(R1)    ;set break bit in xmitter status register
175636  005002                        clr     R2              ;set R2 to the 'null' character
175640  004737  176004                jsr     PC,sndwat       ;send R2 and wait for xmitter ready
175644  103451                        bcs     30$             ;br if time out waiting for 'xmit ready
175646  004737  176004                jsr     PC,sndwat       ;send another null (times length of break
175652  103446                        bcs     30$             ;br if time out waiting for 'xmit ready
175654  105761  000004        10$:    tstb    4(R1)           ;wait for xmitter ready
175660  100375                        bpl     10$             ;(when ready, indicates first null done sendi
175662  005061  000004                clr     4(R1)           ;clear the break bit
175666  012702  000004                mov     #INIT,R2        ;set R2 to tu58 'INIT' character
175672  004737  176004                jsr     PC,sndwat       ;send char in R2 and wait for 'Xmit ready
175676  103434                        bcs     30$             ;br if time out waiting for 'xmit ready
175700  004737  176004                jsr     PC,sndwat       ;send another INIT
175704  103431                        bcs     30$             ;br if time out waiting for 'xmit ready
175706  105761  000004        20$:    tstb    4(R1)           ;wait for xmitter ready to set
175712  100375                        bpl     20$             ;(when ready, indicates first  cont  done sendin
175714  026161  000002  000002        cmp     2(R1),2(R1)     ;read rbuf twice to clear out any junk characters
```

```
                                ;+
                                ;-----------------------------------------------------------------
                                ;
                                ;       Allow up to 2 seconds for TU to respond before
                                ;       rejecting this controller
                                ;
                                ;-----------------------------------------------------------------
                                ;-
175722  012703  000050  PTIME2: mov     #40.,R3         ;set up for 40 delays of 50 MSEC each (=2 seconds)
175726  012702  015034  210:    mov     #6684.,R2       ;R2 gets timing constant
175732  105711          220:    tstb    (R1)            ;got a character yet?           3.14 usec
175734  100403                  bmi     230             ;br if we did                   1.72 usec
175736  077203                  sob     R2,220          ;br until R2 = 0                2.62 usec (total 7.40)
175740  077306                  sob     R3,210          ;do delay 40 times
175742  000412                  br      300             ;take error exit

175744  016102  000002  230:    mov     2(R1),R2        ;get received character to R2
175750  100004                  bpl     240             ;br if no UART errors on received character
175752  112737  000010  000420  movb    #TURCER,TUSHER  ;remember we got a UART error
175760  000406                  br      350             ;take error exit

175762  120227  000020  240:    cmpb    R2,#CONT        ;see if 'TU58 CONTINUE' was received
175766  001404                  beq     400             ;br if continue received
175770  112737  000002  000420  300:    movb    #TUSYNC,TUSHER  ;remember we had a sync failure
175776  022707          350:    cmp     (PC)+,PC        ;set c bit, skip next instruction
176000  000241          400:    clc                     ;clear c bit
176002  000207                  rts     PC              ;return
                                .dsabl  lsb
```

```
                        .sbttl  Send Character to TU58

                ;+
                ;*********************************************************************
                ;
                ;       Send character in R2 to the TU58 specified by address in R1.
                ;       Wait up to 1/2 second for 'Xmit ready' to set before declaring
                ;       a timeout.
                ;
                ;Inputs:
                ;       R1 = Address of a TU58 Receiver Status Register
                ;       R2 = Character to be sent
                ;       SP -> Return address
                ;
                ;Process:
                ;       Wait up to 1/2 second for the 'Xmit Ready  flag to set
                ;       Move R2 to the TU58 'Xmit Buffer' after xmit ready sets
                ;
                ;Outputs:
                ;       C bit is clear (0) if the transmit was successful
                ;       C bit set (1) if the transmit ready flag does not set in time
                ;       R2 = undefined
                ;
                ;*********************************************************************
                ;-
                        .enabl lsb
176004          sndbut: ;+
                        ;---------------------------------------------------------------
                        ;
                        ;       Make sure 'Transmit Ready' is set before transmitting the
                        ;       character in R2.  If 'Transmit Ready' does not set within
                        ;       the timeout period, declare an error. (exit c set)
                        ;
                        ;---------------------------------------------------------------
                        ;-
176004  010246          mov     R2,-(SP)        ;save character to send on stack
176006  005002          clr     R2              ;clear R2 for use as a counter
176010  105761  000004  100$:   tstb    4(R1)           ;test the 'Transmit Ready' bit
176014  100407          bmi     20$             ;br if 'Transmit Ready' is set
176016  005202          inc     R2              ;add one to counter
176020  001373          bne     10$             ;br if counter is not zero
176022  112737  000006  000420  movb    #TUSNTO,TUSER   ;remember we had a transmit timeout
176030  000261          sec                     ;set c bit to indicate time-out
176032  000405          br      30$             ;exit with c set
176034                  20$:    ;+
                        ;---------------------------------------------------------------
                        ;
                        ;       Put the character in R2 into the 'Transmit Buffer'
                        ;
                        ;---------------------------------------------------------------
                        ;-
176034  042716  177400          bic     #177400,(SP)    ;mask out upper 8 bits
176040  011661  000006          mov     (SP),6(R1)      ; 'Transmit Buffer' <- character to send
176044  000241                  clc
176046  012602          30$:    mov     (SP)+,R2        ;restore character to R2
176050  000207                  rts     PC              ;return
                        .dsabl  lsb
```

```
                        .sbttl  Receive a Character from a TU58

                ;+
                ;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
                ;
                ;       Receive one character from a TU58
                ;
                ;inputs:
                ;       R1 = address of TU58 Receiver Status Register
                ;       R2 = SCRATCH
                ;
                ;process:
                ;       Wait up to 40 seconds for a character to arrive.  If a character
                ;       arrives within the time allowed, put character into R2
                ;
                ;outputs:
                ;       C bit is clear if the reception was successful.  R2 = character
                ;       C bit is set if the reception fails, R2 undefined.
                ;
                ;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
                ;-

                        .enabl  lsb
176052          RECMAT: ;+
                        ;--------------------------------------------------------------------
                        ;
                        ;       Create a one word counter on the stack to time the arrival
                        ;       of a character, then begin waiting for the character to arrive.
                        ;       (wait up to 40 seconds before declaring timeout)
                        ;
                        ;--------------------------------------------------------------------
                        ;-
176052  012746  000110  PTIME3: mov     @T50TMO,-(SP)   ;put secondary timer count on stack
176056  005002                clr     R2              ;use R2 as primary timer
176060  105711          100:   tstb    (R1)            ;test 'Receiver Done' bit (bit 7)
176062  100412                bmi     200             ;br if 'Receiver Done' is set (byte is neg)
176064  005202                inc     R2              ;add one to counter
176066  001374                bne     100             ;br if counter is not zero
176070  005316                dec     (SP)            ;subtract one from secondary timer
176072  001372                bne     100             ;br if secondary is not zero yet
176074  112737  000007  000420        movb    @TURCTO,TUSER   ;remember we had a receiver timeout
176102  005726          150:   tst     (SP)+           ;remove counter word from stack
176104  000261                sec                     ;set c bit to indicate error
176106  000412                br      300             ;exit with c set

176110                  200:   ;+
                        ;--------------------------------------------------------------------
                        ;
                        ;       Read the character received to R2 and check for errors
                        ;
                        ;--------------------------------------------------------------------
                        ;-
176110  016102  000002        mov     2(R1),R2        ;R2 <- contents of 'Receiver Buffer'
176114  100004                bpl     250             ;br if no error set in RBUF
176116  112737  000010  000420        movb    @TURCER,TUSER   ;remember we had a UART error
176124  000766                br      150             ;take error exit
```

```
2174 176126  042702  177400        256:    bic     017?400,R2      ;mask character received to 8 bits
2175                                        ;+
2176                                        ;-------------------------------------------------------------
2177                                        ;
2178                                        ;       Return with c bit set (error) or clear (success)
2179                                        ;
2180                                        ;-------------------------------------------------------------
2181                                        ;-
2182 176132  005726                        tst     (SP)+           ;remove counter word from stack
2183                                        ;clc
2184 176134  000207                300:    rts     PC              ;return
2185                                        .dsabl  lsb
```

```
                        .sbttl  Send a TU58 Command Packet
                ;+
                ;*****************************************************************************
                ;
                ;       Send a Command Packet to a TU58
                ;
                ;Inputs:
                ;       R3 = pointer to first byte of packet to be sent(word boundary)
                ;       R2 = SCRATCH
                ;       R1 = pointer to a TU58 Receiver Status Register
                ;       (packet length of 12 bytes, plus 2 byte checksum, is implied)
                ;
                ;Process:
                ;       Transmit 12 bytes, beginning with the byte pointed to by R3.
                ;       Then transmit the 2 byte checksum of the first 12 bytes.
                ;
                ;Outputs:
                ;       C bit clear if the transmission was successful
                ;       C bit set if transmission failed
                ;       R2 = UNDEFINED
                ;       R3 = computed checksum word
                ;       NOTE: R3 is loaded with the checksum merely as a convenient place to
                ;             stick the checksum when clearing off the stack.
                ;
                ;NOTES:
                ;       1) When transmitting, this routine allows 1/2 second for the
                ;          'Xmit Ready' flag to set after transmission.  Failure of
                ;          this flag to set in time will cause an error exit from this
                ;          routine.
                ;
                ;*****************************************************************************
                ;-
                        .enabl  lsb
176136          sendpk: ;+
                        ;-------------------------------------------------------------------
                        ;
                        ;       Initialize checksum word, message word counter, and
                        ;       a one-word transmission temporary location.  All three
                        ;       variables are created on the stack.
                        ;
                        ;-------------------------------------------------------------------
                        ;-

176136  005046          clr     -(SP)           ;create a word for checksum
176140  012746  000006  mov     #6,-(SP)        ;create a counter(12 bytes precede checksum)
176144  005046          clr     -(SP)           ;create a temporary location for transmitter

176146          100:    ;+
                        ;-------------------------------------------------------------------
                        ;
                        ;       Get a word to send
                        ;
                        ;-------------------------------------------------------------------
                        ;-

176146  012316          mov     (R3)+,(SP)      ;put word to send on stack
```

```
                                ;+
                                ;-------------------------------------------------------------
                                ;
                                ;       include word in checksum
                                ;
                                ;-------------------------------------------------------------
                                ;-

                                add     (SP),4(SP)      ;add word to checksum
                                adc     4(SP)           ;include carry-out if C bit is set

                                ;+
                                ;-------------------------------------------------------------
                                ;
                                ;       Send the lower byte of the word on top of stack
                                ;
                                ;-------------------------------------------------------------
                                ;-

                                mov     (SP),R2         ;R2 gets the word being transmitted
                                jsr     PC,sndbat       ;send R2 to transmitter buffer
                                bcs     1000            ;br if transmission failed

                                ;+
                                ;-------------------------------------------------------------
                                ;
                                ;       Send the upper byte of the word on top of stack
                                ;
                                ;-------------------------------------------------------------
                                ;-

                                mov     (SP),R2         ;R2 gets word to send
                                swab    R2              ;swap bytes so high byte in lower byte of R2
                                jsr     PC,sndbat       ;send the lower byte of R2 to xmitter buffer
                                bcs     1000            ;br if transmission failed

                                ;+
                                ;-------------------------------------------------------------
                                ;
                                ;       Check to see if it is time to send the checksum word
                                ;
                                ;-------------------------------------------------------------
                                ;-

                                dec     2(SP)           ;subtract 2 from transmission word counter
                                bgt     10$             ;br if another word to transmit
```

```
                                ;+
                                ;-------------------------------------------------------------
                                ;
                                ;       Send low byte of computed checksum
                                ;
                                ;-------------------------------------------------------------
                                ;-

2299 176210  016602  000004     mov     4(SP),R2        ;R2 gets checksum word
2300 176214  004737  176004     jsr     PC,sndbut       ;transmit the lower byte of R2
2301 176220  103405             bcs     1000            ;br if transmission failed
2302
2303                            ;+
2304                            ;-------------------------------------------------------------
2305                            ;
2306                            ;       Send high byte of computed checksum
2307                            ;
2308                            ;-------------------------------------------------------------
2309                            ;-
2310
2311 176222  016602  000004     mov     4(SP),R2        ;R2 gets checksum word
2312 176226  000302             swab    R2              ;swap bytes so high byte now in low byte of R2
2313 176230  004737  176004     jsr     PC,sndbut       ;transmit lower byte of R2
2314
2315 176234                 1000:  ;+
2316                            ;-------------------------------------------------------------
2317                            ;
2318                            ;       Remove temporary words from stack, then exit with
2319                            ;       c set if error, c clear if no error on transmission
2320                            ;
2321                            ;-------------------------------------------------------------
2322                            ;-
2323
2324 176234  012626             mov     (SP)+,(SP)+     ;pop top two words off stack
2325 176236  012603             mov     (SP)+,R3        ;pop one more word off stack(R3 gets checksum)
2326 176240  000207             rts     PC              ;return
2327
2328                            .dsabl  lsb
```

```
                        .sbttl  Receive a Command or Data Packet from the TU58
                ;+
                ;*****************************************************************************
                ;
                ;       Receive a Command or Data Packet from a TU58
                ;
                ;Inputs:
                ;       R1 = address of TU58 Receiver Status Register
                ;       R3 = pointer to a packet buffer
                ;       R2 = pointer to data storage area (only used for incoming data packets)
                ;
                ;Process:
                ;       if (first byte = 'Data Packet')
                ;       then ( input data bytes to memory )
                ;       else ( input 'command packet' to the packet buffer )
                ;
                ;       Compute checksum
                ;
                ;Outputs:
                ;       R1 = UNCHANGED
                ;       R2 = pointer to next data storage location
                ;       R3 = UNDEFINED
                ;       R4 and R5 are PRESERVED
                ;
                ;       C bit clear if reception was successful
                ;       C bit set if reception failed
                ;
                ;NOTES:
                ;       1) command packets are just inputted to the packet buffer, and the
                ;          checksum is verified before returning to the calling process.
                ;
                ;       2) The first two bytes of a 'Data' packet are inputted to the packet
                ;          buffer, but the data is stored on the fly
                ;
                ;       3) This routine will wait a maximum of 40 seconds between characters
                ;          sent from the TU58.  Time-out while waiting for 'Receiver Done' to
                ;          set causes this routine to exit with the C bit set (error indication).
                ;
                ;*****************************************************************************
                ;-
                        .enabl lsb
176242          recvpk: ;+
                        ;-------------------------------------------------------------- -----
                        ;
                        ;       Preserve some registers, and create some temporary
                        ;       storage locations on the stack
                        ;
                        ;--------------------------------------------------------------------
                        ;-

176242  010046                mov     R0,-(SP)        ;preserve R0
176244  010546                mov     R5,-(SP)        ;preserve R5
176246  005046                clr     -(SP)           ;create a checksum word
176250  005046                clr     -(SP)           ;create a message word counter
176252  005046                clr     -(SP)           ;create a temporary for the receiver process
176254  010200                mov     R2,R0           ;put the data storage pointer in R0
```

RECEIVE A COMMAND OR DATA PACKET FROM THE TU58

```
                        ;+
                        ;-------------------------------------------------------------
                        ;
                        ;       NOTE: CURRENT STACK PICTURE
                        ;
                        ;       +10     return address
                        ;       +8      saved R5
                        ;       +6      saved R0
                        ;       +4      checksum word
                        ;       +2      message word counter
                        ;       SP -)   receiver temporary storage word
                        ;
                        ;-------------------------------------------------------------
                        ;-


                        ;+
                        ;-------------------------------------------------------------
                        ;
                        ;       input a 'low' byte
                        ;
                        ;-------------------------------------------------------------
                        ;-

2410 176256  004737  176052     jsr     PC,recmat       ;receive a character in R2
2411 176262  103521             bcs     40$             ;br if reception error
2412 176264  010216             mov     R2,(SP)         ;save inputted byte on stack
2413
                        ;+
                        ;-------------------------------------------------------------
                        ;
                        ;       input a 'high' byte
                        ;
                        ;-------------------------------------------------------------
                        ;-

2422 176266  004737  176052     jsr     PC,recmat       ;receive a character in R2
2423 176272  103515             bcs     40$             ;br if reception failed
2424 176274  000316             swab    (SP)            ;swap bytes on stack
2425 176276  060216             add     R2,(SP)         ;include byte just inputted
2426 176300  000316             swab    (SP)            ;put bytes in proper order
2427
                        ;+
                        ;-------------------------------------------------------------
                        ;
                        ;       check for a 'command' or 'data' packet by looking
                        ;       at the first byte received
                        ;
                        ;-------------------------------------------------------------
                        ;-

2437 176302  011623             mov     (SP),(R3)+      ;store first two bytes of packet in buffer
2438 176304  011666  000004     mov     (SP),4(SP)      ;include first 2 bytes in checksum
2439 176310  121627  000001     cmpb    (SP),#data      ;check for a data packet code
2440 176314  001456             beq     20$             ;br if incoming packet is a 'data' packet
```

```
                                        ;+
                                        ;-------------------------------------------------------------
                                        ;
                                        ;       receive the rest of a 'command' packet, and store it
                                        ;       in the packet buffer.  Receive 8 more bytes of command,
                                        ;       and 2 bytes of checksum
                                        ;
                                        ;-------------------------------------------------------------
                                        ;-

176316  005266  000002                  inc     2(SP)           ;bump word counter

176322                          100:    ;+
                                        ;-------------------------------------------------------------
                                        ;
                                        ;       get a 'low' byte
                                        ;
                                        ;-------------------------------------------------------------
                                        ;-

176322  004737  176052                  jsr     PC,recmat       ;receive a character in R2
176326  103477                          bcs     400             ;br if reception failed
176330  010216                          mov     R2,(SP)         ;store low byte on stack

                                        ;+
                                        ;-------------------------------------------------------------
                                        ;
                                        ;       get a 'high' byte
                                        ;
                                        ;-------------------------------------------------------------
                                        ;-

176332  004737  176052                  jsr     PC,recmat       ;receive a character in R2
176336  103473                          bcs     400             ;br if reception failed
176340  000316                          swab    (SP)            ;swap bytes on stack
176342  060216                          add     R2,(SP)         ;include high byte just received
176344  000316                          swab    (SP)            ;put bytes in proper order

                                        ;+
                                        ;-------------------------------------------------------------
                                        ;
                                        ;       include word just received in checksum, add one to the
                                        ;       word counter, and store word to packet buffer
                                        ;
                                        ;-------------------------------------------------------------
                                        ;-

176346  061666  000004                  add     (SP),4(SP)      ;add word to checksum
176352  005566  000004                  adc     4(SP)           ;include possible carry in checksum
176356  005266  000002                  inc     2(SP)           ;add one to word counter
176362  011623                          mov     (SP),(R3)+      ;store word to packet buffer
```

RECEIVE A COMMAND OR DATA PACKET FROM THE TU58

```
2494                                    ;+
2495                                    ;-----------------------------------------------------------------------
2496                                    ;
2497                                    ;       check to see if a complete 'command packet' (minus checksum)
2498                                    ;       has been received (6 words)
2499                                    ;
2500                                    ;-----------------------------------------------------------------------
2501                                    ;-
2502
2503 176364  026627  000002  000006          cmp     2(SP),#6        ;check for 6 words inputted
2504 176372  002753                          blt     10$             ;br if another word to receive
2505
2506 176374                          15$:    ;+
2507                                    ;-----------------------------------------------------------------------
2508                                    ;
2509                                    ;       Get low byte of checksum
2510                                    ;
2511                                    ;-----------------------------------------------------------------------
2512                                    ;-
2513
2514 176374  004737  176852                  jsr     PC,recmat       ;receive a character to R2
2515 176400  103452                          bcs     40$             ;br if reception failed
2516 176402  010216                          mov     R2,(SP)         ;save low byte on stack
2517
2518                                    ;+
2519                                    ;-----------------------------------------------------------------------
2520                                    ;
2521                                    ;       Get high byte of checksum
2522                                    ;
2523                                    ;-----------------------------------------------------------------------
2524                                    ;-
2525
2526 176404  004737  176852                  jsr     PC,recmat       ;receive a character in R2
2527 176410  103446                          bcs     40$             ;br if reception failed
2528
2529                                    ;+
2530                                    ;-----------------------------------------------------------------------
2531                                    ;
2532                                    ;       Assemble checksum word in R2, and compare computed checksum
2533                                    ;       with the checksum in R2.  Exit with c bit set if checksums
2534                                    ;       do not agree.
2535                                    ;
2536                                    ;-----------------------------------------------------------------------
2537                                    ;-
2538
2539 176412  000302                          swab    R2              ;put high byte where it belongs
2540 176414  061602                          add     (SP),R2         ;include low byte
2541 176416  020266  000004                  cmp     R2,4(SP)        ;compare R2 to computed checksum
2542 176422  001411                          beq     10$             ;br if checksums agree
2543 176424  112737  000005  000420          movb    #TUCKSM,TUSER   ;remember we had a checksum error
2544 176432  010237  000416                  mov     R2,CKSMAC       ;save actual checksum from TU58
2545 176436  016637  000004  000414          mov     4(SP),CKSMEX    ;save expected checksum we computed
2546 176444  000430                          br      40$             ;take error exit
2547
2548 176446  000241                  10$:    clc                     ;clear c bit to indicate success
2549 176450  000427                          br      50$             ;clean stack and exit with c clear
```

RECEIVE A COMMAND OR DATA PACKET FROM THE TU58

```
2551 176452                    200:    ;+
2552                                   ;-------------------------------------------------------------
2553                                   ;
2554                                   ;       Data Packet is incoming.  Two bytes received so far are
2555                                   ;       on the top of the stack.  low byte is 'data packet' code.
2556                                   ;       high byte is the byte count for the data arriving next.
2557                                   ;       Put byte count in lower byte, and clear upper byte.
2558                                   ;
2559                                   ;-------------------------------------------------------------
2560                                   ;-
2561
2562 176452  042716  000377            bic     #377,(SP)       ;clear out 'data packet' code
2563 176456  000316                    swab    (SP)            ;put byte count into lower byte
2564 176460  011605                    mov     (SP),R5         ;put byte count into R5
2565
2566 176462                    300:    ;+
2567                                   ;-------------------------------------------------------------
2568                                   ;
2569                                   ;       Get a 'low' byte
2570                                   ;
2571                                   ;-------------------------------------------------------------
2572                                   ;-
2573
2574 176462  004737  176652            jsr     PC,recmat       ;receive a character in R2
2575 176466  103417                    bcs     400             ;br if reception failed
2576 176470  010216                    mov     R2,(SP)         ;save low byte
2577
2578                                   ;+
2579                                   ;-------------------------------------------------------------
2580                                   ;
2581                                   ;       Get a 'high' byte
2582                                   ;
2583                                   ;-------------------------------------------------------------
2584                                   ;-
2585
2586 176472  004737  176652            jsr     PC,recmat       ;receive a character in R2
2587 176476  103413                    bcs     400             ;br if reception failed
2588 176500  000302                    swab    R2              ;move byte to proper position
2589 176502  061602                    add     (SP),R2         ;include low byte
2590
2591                                   ;+
2592                                   ;-------------------------------------------------------------
2593                                   ;
2594                                   ;       include word in checksum
2595                                   ;
2596                                   ;-------------------------------------------------------------
2597                                   ;-
2598
2599 176504  060266  000004            add     R2,4(SP)        ;add inputted word to computed checksum
2600 176510  005566  000004            adc     4(SP)           ;include possible carry in checksum
```

```
2602                                    ;+
2603                                    ;-------------------------------------------------------------
2604                                    ;
2605                                    ;       Store inputted word via the pointer in R0
2606                                    ;
2607                                    ;-------------------------------------------------------------
2608                                    ;-
2609
2610 176514  010220                     mov     R2,(R0)+
2611
2612                                    ;+
2613                                    ;-------------------------------------------------------------
2614                                    ;
2615                                    ;       test for byte count exhausted.  If not exhausted, get
2616                                    ;       another word, otherwise input checksum word
2617                                    ;
2618                                    ;-------------------------------------------------------------
2619                                    ;-
2620
2621 176516  162705  000002             sub     #2,R5           ;byte count gets byte count minus 2
2622 176522  003357                     bgt     30$             ;br if another word to input
2623 176524  000723                     br      15$             ;go get checksum
2624
2625 176526                      40$:   ;+
2626                                    ;-------------------------------------------------------------
2627                                    ;
2628                                    ;       error, set c bit and exit
2629                                    ;
2630                                    ;-------------------------------------------------------------
2631                                    ;-
2632
2633 176526  000261                     sec                     ;set c bit to indicate error
2634
2635 176530                      50$:   ;+
2636                                    ;-------------------------------------------------------------
2637                                    ;
2638                                    ;       clean up stack and exit
2639                                    ;
2640                                    ;-------------------------------------------------------------
2641                                    ;-
2642
2643 176530  010002                     mov     R0,R2           ;restore the data pointer to R2
2644 176532  012626                     mov     (SP)+,(SP)+     ;pop off two words (leaving c bit unaffected)
2645 176534  012603                     mov     (SP)+,R3        ;pop off another word (leaving c bit unaffected)
2646 176536  012605                     mov     (SP)+,R5        ;restore R5
2647 176540  012600                     mov     (SP)+,R0        ;restore R0
2648 176542  000207                     rts     PC              ;return
2649                                    .dsabl lsb
```

```
                        .sbttl  Routine to save TU58 error reasons
                    ;+
                    ;*****************************************************************************
                    ;
                    ;       Save TU58 error reasons
                    ;
                    ;Inputs:  'TUERPT' = points to error word for unit 0 of controller we just tried
                    ;                    to boot from.
                    ;         R4 = unit # we tried to boot from (0 or 1)
                    ;         'TUSHER' = error reason generated by ROM software
                    ;         'TUHWER' = error code from TU58 end packet (0 if none)
                    ;
                    ;process: 1) select error word to use.
                    ;               IF ( R4 = 0 )
                    ;               THEN ( error word = contents of TUERPT )
                    ;               ELSE ( error word = (contents of TUERPT) + 2)
                    ;         2) Upper byte of error word gets low byte of TUHWER
                    ;            Lower byte of error word gets low byte of TUSHER
                    ;
                    ;outputs: All registers are preserved.
                    ;
                    ;*****************************************************************************
                    ;-
                            .enabl  lsb
2676 176544  010046 SAVEER: mov     R0,-(SP)        ;preserve R0
2677 176546  013700  000422         mov     TUERPT,R0       ;R0 gets pointer to error word for unit 0
2678 176552  005704                 tst     R4              ;is unit = 0 ?
2679 176554  001401                 beq     10$             ;br if yes
2680 176556  005720                 tst     (R0)+           ;add 2 to R0
2681 176560  013710  000420 10$:    mov     TUERR0,(R0)     ;save HW/SW error codes in table
2682 176564  012600                 mov     (SP)+,R0        ;restore R0
2683 176566  000207                 rts     PC              ;return
2684                                .dsabl  lsb
```

```
                        .sbttl  Moving Inversions Test Description
                ;+
                ;*****************************************************************************
                ;       Modified Moving Inversions RAM Test for PDP-11's
                ;
                ;TEST STRATEGY: Starting with the memory cleared to zeroes, make 'n' passes (*)
                ;       from lowest address to highest address.  On each pass, every location is
                ;       read to verify that the last pattern was stored correctly, then the data
                ;       in the location is rewritten to contain an additional 'one'.  This is
                ;       continued until each location contains all ones.  Now that the memory
                ;       contains all 'ones', make 'n' passes thru the memory from highest address
                ;       to lowest, this time read each location and verify that the previous
                ;       pattern is correct, replace a single '1' with a '0', and reread each
                ;       location to insure that the new pattern was stored. When finished with 16
                ;       passes, the memory contains all zeroes again.
                ;
                ; (*) 'n' = number of binary bits per location
                ;
                ;Test Steps:
                ;       1)  Clear memory to all 'zeroes'.
                ;       2)  Starting at the LOWEST address, and working UP to the highest, read
                ;           a location and verify that the previous data was stored, replace
                ;           the location's data with a single additional 'one' bit, and reread
                ;           the location to insure that the new pattern was stored correctly.
                ;           This sequence is repeated until, on the final pass, the memory
                ;           contains all ones. (16 passes thru the memory).
                ;       3)  Starting now at the HIGHEST memory address and working DOWN, read
                ;           each location and verify that the previous pattern is read cor-
                ;           rectly, rewrite the location's data to contain a single additional
                ;           'zero' bit, then reread the location to insure that the new pattern
                ;           was stored correctly.  Continue until the memory contains all zeroes.
                ;           (After the 16th pass.)
                ;       4)  End of test.
                ;
                ;FAILURE INDICATIONS: On each of the steps described above as a 'check', failure
                ;       to find the expected value is an error.
                ;
                ;*****************************************************************************
                ;-
```

```
                                .sbttl  Moving Inversions Test

                        ;+
                        ;*********************************************************************
                        ;INPUTS:
                        ;       R0,R1,R2,R3 are SCRATCH
                        ;       R4 = Base address of the section of memory under test
                        ;       R5 = Last address of the section of memory under test
                        ;       R6 = Stack Pointer. (WARNING: do not push more than 2 words)
                        ;
                        ;OUTPUTS:
                        ;       C Bit set if test FAILED
                        ;               R0 = failing address
                        ;               R4,R5 unchanged
                        ;       C bit clear if test PASSED
                        ;               R4,R5 unchanged
                        ;               R0 thru R3 undefined
                        ;
                        ;REGISTER USAGE:
                        ;       R0 = Address under test
                        ;       R1 = Data Pattern to check for
                        ;       R2 = Data Pattern to store
                        ;       R4 = Base Address of memory under test
                        ;       R5 = Last Address of memory under test
                        ;       R6 = Stack pointer (WARNING: do not push more than 2 words)
                        ;
                        ;*********************************************************************
                        ;-
176570                  MOVINV: ;+
                                ;*********************************************************************
                                ;       STEP 1: Clear all memory locations to zeroes.
                                ;*********************************************************************
                                ;-

176570  010400                  MOV     R4,R0           ;SET R0 TO BASE ADDRESS OF MEMORY TO TEST
176572  005020          100:    CLR     (R0)+           ;CLEAR A LOCATION(WORD), BUMP ADDRESS BY 2
176574  020005                  CMP     R0,R5           ;CHECK FOR THE LAST ADDRESS CLEARED
176576  101775                  BLOS    100             ;BR IF MORE TO CLEAR
                                ;+
                                ;*********************************************************************
                                ; STEP 2: Starting at the LOWEST address, and working UP to the highest,
                                ;       read each location and verify that the previous pattern is read
                                ;       correctly, rewrite the location's data to contain a single
                                ;       additional one, then reread the location to verify that the new
                                ;       pattern was stored correctly.  This sequence is repeated 16
                                ;       times, until the memory contains all 'ones'.
                                ;*********************************************************************
                                ;-

176600  005002                  CLR     R2              ;INITIALIZE 'PATTERN TO STORE'
176602  010201          200:    MOV     R2,R1           ;REPLACE THE LAST PATTERN WITH THE NEXT TO CHECK FOR
176604  006302                  ASL     R2              ;SET ONE MORE 'ONE' BIT IN R2(PATTERN TO STORE)
176606  005202                  INC     R2              ;(USING 'INC' VS. 'SEC' SAVES TIME)
176610  010400                  MOV     R4,R0           ;SET R0 TO BASE ADDRESS OF MEMORY TO TEST
176612  020110          300:    CMP     R1,(R0)         ;CHECK FOR LAST PATTERN STORED SUCCESSFULLY
176614  001031                  bne     1900            ;br if last pattern is not correct
176616  010210                  MOV     R2,(R0)         ;STORE NEXT PATTERN
```

```
 2783 176620  020210                    CMP     R2,(R0)         ;CHECK FOR NEW PATTERN
 2784 176622  001026                    bne     1906            ;br if new pattern NOT stored correctly
 2785 176624  062700  000002            add     #2,R0           ;bump test address by 2
 2786 176630  020005                    CMP     R0,R5           ;CHECK FOR LAST ADDRESS REACHED
 2787 176632  101767                    BLOS    30$             ;BRANCH IF MORE ADDRESSES TO TEST
 2788 176634  005702                    tst     R2              ;when R2 = 177777 goto next step
 2789 176636  100361                    bpl     20$             ;br if more passes to make
 2790                                    ;+
 2791                                    ;*********************************************************************
 2792                                    ;  STEP 3: Starting at the HIGHEST memory address, and working DOWN to
 2793                                    ;       the lowest, read each location and verify that the previous
 2794                                    ;       pattern can be read correctly, rewrite the location's data to
 2795                                    ;       contain a single additional 'zero' bit, then reread the location
 2796                                    ;       to check that the new pattern was stored correctly.  Repeat this
 2797                                    ;       sequence until (on the 16th pass) the memory contains all zeroes.
 2798                                    ;*********************************************************************
 2799                                    ;-
 2800
 2801                                    ;register state on initial entry to this step from previous step.
 2802                                    ;       R2 = 177777,  R1 = 077777
 2803
 2804 176640  010201            600:    MOV     R2,R1           ;SET R1 TO  PATTERN TO CHECK FOR
 2805 176642  005700                    tst     R0              ;(fastest way to clear the 'c' bit
 2806 176644  006002                    ror     R2              ;add another zero to pattern in R2
 2807 176646  010500                    MOV     R5,R0           ;SET R0 TO HIGHEST MEMORY ADDRESS
 2808 176650  062700  000002            ADD     #2,R0           ;ADJUST ADDRESS FIRST TIME SO WE CAN USE DECREMENT INITIAL
LY
 2809 176654  020140            700:    CMP     R1,-(R0)        ;CHECK FOR LAST PATTERN, DECREMENT ADDRESS BY 2
 2810 176656  001010                    bne     1906            ;br if last pattern not stored correctly
 2811 176660  010210                    MOV     R2,(R0)         ;STORE NEXT PATTERN (ONE ADDITIONAL 'ZERO-BIT'
 2812 176662  020210                    CMP     R2,(R0)         ;CHECK FOR PROPER STORAGE OF NEW PATTERN
 2813 176664  001005                    bne     1906            ;br if new pattern NOT stored correctly
 2814 176666  020004                    CMP     R0,R4           ;CHECK FOR LOWEST MEMORY ADDRESS REACHED
 2815 176670  101371                    BHI     70$             ;BR IF STILL NOT AT LOWEST ADDRESS
 2816 176672  005702                    tst     R2              ;when R2 = 0, the test is done
 2817                                    ;clc                    ;(side effect of clc instruction)
 2818 176674  001361                    BNE     60$             ;BR IF ANOTHER PASS THRU MEMORY IS REQUIRED
 2819                                    ;+
 2820                                    ;*********************************************************************
 2821                                    ;
 2822                                    ;       Exit
 2823                                    ;
 2824                                    ;*********************************************************************
 2825                                    ;-
 2826
 2827 176676  000207                    rts     PC              ;exit with c bit clear
 2828
 2829 176700                    1906:   ;(ERROR EXIT)
 2830 176700  000261                    sec
 2831 176702  000207                    rts     PC              ;return with c bit set
```

```
                        .sbttl  Permanent Storage

                        ; 'DIAGNOSE' command packet for TU58
176704  002     diagpk: .byte   2               ; flag byte = 'command'
176705  012             .byte   10.             ; command byte count = ten
176706  007             .byte   diagno          ; op-code = 'diagnose'
176707  000             .byte   0               ; modifier byte = zero
176710  000             .byte   0               ; unit # = zero
176711  000             .byte   0               ; switches = zero
176712  000             .byte   0               ; seq # (low)  = zero
176713  000             .byte   0               ; seq # (high) = zero
176714  000             .byte   0               ; data byte count (low)  = zero
176715  000             .byte   0               ; data byte count (high) = zero
```

```
                        ;*
                        ;**********************************************************************
                        ;
                        ;       END OF P.10C ROM BOOTSTRAP PROGRAM
                        ;
                        ;**********************************************************************
                        ;-

                                ;+
                                ;--------------------------------------------------------------
                                ;
                                ;       'REMAIN' gets defined to be equal to the # of bytes left
                                ;       in the ROM (if positive), or to the # of bytes exceeded
                                ;       (if negative).
                                ;
                                ;       if (ROM size is exceeded by this program)
                                ;       then (issue assembly error)
                                ;
                                ;--------------------------------------------------------------
                                ;-

000062                          remain=lasadd-.+1       ;compute the # of bytes remaining or exceeded

                                .if lt remain-2
                                .error  ;ROM size exceeded by remain bytes
                                .iff
                                .=LASADD-1
176776  000001                  .word   PBMEDT*256.!PBMVER      ;low byte = ROM version, High byte = ROM edit
                                .endc

173000                          .end    coldbt
```

SYMBOL TABLE

```
ALLUMP= 000037      HIPTST= 000040      KB8THR= 000054      RECVPK  176242      TUDMTO= 000006
ALTENT  173036      INIUMP= 000020      KB8TYP= 000043      RECWAT  176052      TU58FL= 000023
ANYFAL  174572      INIT  = 000004      KB8URS= 000044      REMAIN= 000062      TU58PC  173030
BADMEM  174566      JMPTAR  173614      KB8.PF= 000000      RESETS= 000005      TU58RD  173020
BASADD= 173000      JMPTR1  173630      KB8.SZ= 000140      SAVERR  176544      TU58SN  173022
BLKLMP= 000004      JMPTR2  173622      LASADD= 176777      SELP0 = 000400      TU58SY  173016
BLKSH = 002000      JSRTST  174002      LCKBIT= 000001      SELP1 = 001000      TS8FAL  175104
BTYPE2  173010      KGD0OG= 000001      LEDBIT= 000010      SENDPK  176136      TS8REA  175442
CKSMAC= 000416      KGD0NF= 000000      LODSIZ= 000010      SL0ER0= 000410      TS8SST  175226
CKSMEX= 000414      KGD0UK= 001000      LOPTST= 000020      SL0ER1= 000412      TS8SYC  175630
CLRBIT= 000100      KGD0UP= 000400      LS    = 065535      SL1ER0= 000400      TS8TMO= 000110
CMND  = 000002      KGP0CI= 000001      MEMDON  174562      SL1ER1= 000402      TS8TST  174664
CNTIE = 000002      KGP0PI= 000277      MEMERR  174514      SL2ER0= 000404      VECSIZ= 004000
COLDBT  173000      KGP0PS= 000076      MEMFAL= 000022      SL2ER1= 000406      WRITE = 000003
CONT  = 000020      KGP0SD= 000002      MEMRTY  174506      SNDWAT  176004      $GLOBL= 000000
CPUFAL  174260      KGP0ST= 000203      MEMTST  174354      SOBBAK  173566      $KPAR0= 172340
CSRLST  175074      KGT0CI= 000001      MIMEMT  173024      SWPBNK= 002000      $KPAR4= ****** GX
DATA  = 000001      KGT0MK= 000077      MLBUSN= 000522      SWPBRD= 004000      $KPAR5= ****** GX
DIAGNO= 000007      KGT0PI= 000077      MODDAT  173724      SWTTST  174372      $KPAR6= ****** GX
DIBPAK  176704      KGT0PS= 000076      MODOT1  173730      TERMEN= 100000      $KPAR7= ****** GX
ENDARD= 000200      KGT0SD= 000002      MOVINV  176570      TESTNO= 172340      $PDRSR0= ****** GX
ENDP  = 000100      KGT0ST= 000003      NXMBIT= 000004      TRNSFR  175112      $PIOCS= ****** GX
ERRRPT  173026      KGV0CV= 000377      PARSIZ= 040000      TUCKSM= 000005      $SL0RS= ****** GX
FLTLMP= 000010      KGV0SV= 177400      PARTST  174436      TUERPT= 000422      $SL1RS= ****** GX
FLTSW = 004000      KGV0AS= 000000      PDMEDT= 000000      TUERRO= 000420      $SL2RS= ****** GX
F11ADC  174012      KGV0MS= 000052      PDMVER= 000001      TUMAER= 000421      $SMCSR= ****** GX
F11ADD  173636      KGV0CA= 000060      PIOFAL= 000021      TUNOM = 000001      $TUBRK= 000001
F11DCN  174266      KGV0MP= 000056      POSITN= 000005      TUPROT= 000011      $$END0= 000140
F11MTP  173734      KGV0SA= 000104      PTIME0  174626      TURCER= 000010      $$FLGS= 000000
F11TST  173032      KGVFBU= 000050      PTIME1  174646      TURCTO= 000007      $$PC0$= 001001
HALMP = 000001      KGVFCD= 000046      PTIME2  175722      TUSLFT= 000003      $$VALS= 000001
HMEM  = 000400      KGVLSA= 000140      PTIME3  176052      TUSOFT= 000004      $$VALU= 177777
HOLMP = 000002      KGVSLT= 000042      READ  = 000002      TUSWER= 000420      ..Y   = 177400
HRSH  = 001000      KGVTLK= 000040      READBT  175362      TUSY    000002
```

```
. ABS.  177000    000
        000000    001
ERRORS DETECTED:  0

VIRTUAL MEMORY USED:  11696 WORDS  ( 46 PAGES)
DYNAMIC MEMORY:  19740 WORDS  ( 75 PAGES)
ELAPSED TIME:  00:01:07
BOOBOJ:PIBOOT,BOOLST:PIBOOT/CR/-SP=BOOPLB:MSCLIB/ML,BOOMAC:PIBOOT.MAC/EN:LC
```

SYMBOL	VALUE	REFERENCES								
ALLUMP	= 000037	010-459	27-1314	27-1327						
ALTEXT	173836	12-524	014-615							
ANYFAL	174572	12-543	22-1085	027-1290	31-1556					
BADMEM	174566	25-1219	26-1273	027-1281						
BASADD	= 173800	07-299	7-334							
BLKUMP	= 000004	010-456	10-459	27-1303						
BLKSM	= 002000	010-451								
BTYPE2	173810	012-523								
CKSMAC	= 000416	07-320	7-329	049-2544						
CKSMEX	= 000414	07-327	7-328	049-2545						
CLKBIT	= 000100	09-406								
CMND	= 000002	00-388	30-1913							
CNTIE	= 000002	09-411								
COLDBT	173800	012-503	57-2000							
CONT	= 000020	00-382	40-2051							
CPUFAL	174260	20-989	20-990	20-992	20-995	20-996	20-999	20-1001	20-1004	20-1006
		21-1022	21-1033	21-1035	21-1037	21-1039	21-1042	22-1056	22-1057	22-1058
		22-1060	22-1062	22-1063	22-1065	22-1066	022-1070	24-1165		
CSRLST	175074	30-1455	031-1535							
DATA	= 000001	00-381	30-1963	47-2439						
DIAGNO	= 000007	00-387	56-2030							
DISPAK	176704	35-1766	056-2036							
DMABIT	= 000200	09-405								
DMP	= 000100	00-388	30-1963							
ERRPT	173826	012-543								
FLTUMP	= 000010	010-455	10-459	27-1303	27-1304					
FLTSM	= 004000	010-450	27-1305							
F11ABC	174012	19-964	020-971							
F11ADD	173636	17-876	018-881							
F11DON	174266	22-1067	022-1087							
F11HTP	173734	18-912	019-920							
F11TST	173832	12-512	014-606							
HALUMP	= 000001	010-458	10-459	27-1303						
HASM	= 000400	010-453								
HBLUMP	= 000002	010-457	10-459	27-1303						
HBSM	= 001000	010-452								
HIPTST	= 000040	09-407								
INILUMP	= 000020	010-454	10-459	22-1095						
INIT	= 000004	00-379	39-2021							
JMPTAB	173614	17-872	017-874							
JMPTR1	173630	17-874	017-878							
JMPTR2	173622	017-876	17-878							
JSRTST	174002	19-962	019-967							
LASADD	= 176777	07-300	57-2060	57-2073						
LCKBIT	= 000001	09-412								
LEDBIT	= 000010	09-409								
LODSIZ	= 000010	00-356	33-1632	36-1847						
LOPTST	= 000020	09-408								
MEMDON	174562	26-1247	026-1276							
MEMERR	174514	26-1240	26-1242	026-1263						
MEMFAL	= 000022	00-367	27-1289							
MEMRTY	174506	026-1246	26-1274							

SYMBOL	VALUE	REFERENCES							
MEMTST	174354	025-1176	25-1216	25-1222					
MIMEMT	173024	012-542							
MLBVSN	= 000522	02-69							
MODDAT	173724	10-093	10-097	10-900	10-905	010-915	10-915	10-916	
MODOT1	173730	010-917	10-918						
MOVINU	176570	12-542	25-1191	26-1246	055-2754				
NOWAIT	= 000004	09-410							
PARSIZ	= 040000	00-354	26-1245	26-1271	29-1423	35-1779	36-1843		
PARTST	174436	25-1192	026-1225						
PDMEDT	= 000000	02-27	57-2002						
PDMVER	= 000001	02-27	57-2002						
PIDFAL	= 000021	00-366	22-1004						
POSITN	= 000005	00-386							
PTIME0	174626	027-1316							
PTIME1	174646	027-1320							
PTIME2	175722	040-2030							
PTIME3	176052	042-2149							
READ	= 000002	00-384	30-1915						
READBT	175362	29-1450	30-1505	036-1834					
RECVPK	176242	12-544	35-1781	30-1951	046-2371				
RECWAT	176052	042-2140	47-2410	47-2422	48-2462	48-2474	49-2514	49-2526	50-2574	50-2586
REMAIN	= 000062	057-2060	57-2070						
RESETS	= 000005	09-440	14-615						
SAVERR	176544	30-1489	30-1507	053-2676					
SECNDP	= ******	2-27	14-620	57-2075					
SELP0	= 000400	09-404							
SELP1	= 001000	09-403							
SENDPK	176136	12-541	35-1767	30-1940	043-2220				
SL0ER0	= 000410	07-323	7-324						
SL0ER1	= 000412	07-324	7-327						
SL1ER0	= 000400	07-319	7-320	30-1456					
SL1ER1	= 000402	07-320	7-321						
SL2ER0	= 000404	07-321	7-322						
SL2ER1	= 000406	07-322	7-323						
SNDWAT	176004	39-2014	39-2016	39-2022	39-2024	041-2085	44-2264	44-2277	45-2300	45-2313
SOBUMK	173566	017-864	17-866						
SWPBMK	= 002000	09-402	24-1161	24-1164	24-1168	24-1170	24-1171	25-1213	25-1215	25-1220
SWPBUD	= 004000	09-401	24-1161	24-1162	24-1164	24-1167	24-1168	24-1171	25-1210	25-1221
SWTTST	174372	025-1193							
TERMEN	= 100000	09-400							
TESTMD	= 172340	07-302	*14-616	*24-1159	*25-1175	*26-1230	*29-1412	*33-1614	
TRNSFR	175112	29-1451	31-1531	033-1685					
TUCKSM	= 000005	09-426	49-2543						
TUERPT	= 000422	07-332	*30-1456	*30-1472	53-2677				
TUERRD	= 000420	07-329	7-330	7-331	7-332	*30-1471	*30-1504	53-2681	
TUMAER	= 000421	07-331	*35-1786	*30-1973					
TUMOM	= 000001	09-422	35-1800						
TUPROT	= 000011	09-430	30-1967						
TURCER	= 000010	09-429	40-2040	42-2171					
TURCTO	= 000007	09-428	42-2157						
TUSLFT	= 000003	09-424	35-1787						
TUSOFT	= 000004	09-425	37-1864						

SYMBOL	VALUE		REFERENCES								
TUSNER	= 000420		07-330	#35-1707	#35-1800	#37-1864	#38-1967	#40-2040	#40-2053	#41-2100	#42-2157
			#42-2171	#49-2543							
TUSYNC	= 000002		09-423	40-2053							
TUSXTO	= 000006		09-427	41-2100							
TUSXFL	= 000023		08-360	31-1555							
TUSXRC	173030		012-544								
TUSXRD	173020		012-540								
TUSXSN	173022		012-541								
TUSXSY	173016		012-539								
TSXFAL	175104		30-1474	031-1542							
TSXREA	175442		12-540	36-1849	030-1896						
TSXSST	175226		29-1444	30-1487	035-1727						
TSXSYC	175630		12-539	31-1520	35-1752	039-2000					
TSXTMO	= 000110		08-353	42-2149							
TSXTST	174664		26-1277	029-1412							
VECSIZ	= 004000		08-355	25-1189	26-1239	26-1244	26-1245	26-1263			
WRITE	= 000003		08-305								
$XPARC	= 172340		07-301	7-302							
$XPAR4	= ******	GX	#21-1023	#21-1031	21-1038	21-1041					
$XPAR5	= ******	GX	19-960	20-983	22-1052	25-1211	29-1436	#33-1615	#33-1616	34-1667	34-1694
$XPAR6	= ******	GX	29-1438	29-1443	#33-1617	34-1668					
$XPAR7	= ******	GX	29-1440								
$PISR0	= ******	GX	#19-959								
$PIOCS	= ******	GX	24-1160	25-1212							
$SLORS	= ******	GX	31-1537								
$SLIRS	= ******	GX	31-1535								
$SL2RS	= ******	GX	31-1536								
$SXCSR	= ******	GX	#22-1095	27-1382							
$TUBKK	= 000001		08-309	39-2012							

MACRO NAME REFERENCES

ASSUME	02-69	
DMACC0	02-69	
DMASK0	02-69	
DUPMY0	02-69	
BUMP0	02-69	
FAULT0	02-69	
MMINIT	02-60	2-69
IDENT	02-2	2-27
IDENT1	02-12	2-27
KB0	02-60	2-70
MLBVSN	02-69	2-69
NOTYET	02-69	
PCALL	02-69	
POP	02-69	
PSECT	02-69	
PUSH	02-69	
RESET	02-69	
SAVAL0	02-69	
SAVR20	02-69	
SAVR30	02-69	
SAVR40	02-69	
SAVR50	02-69	
SPL	02-69	
UONULL	02-70	
.ADDR.	02-69	
.BLKB2	02-69	
.BLKW2	02-69	
.FROM.	02-69	
.LOCK.	02-69	
.REG.	02-69	
.TO.	02-69	

CHAPTER 4
HSC70 BOOT PROM LISTINGS

This chapter contains the Boot PROM listings for the HSC70 The page numbers for this chapter are in the upper right hand corner of the listings

Table of Contents

2- 1 ***** MBC70 ROM Bootstrap version X03-01 *****
4- 1 Useful Information
5- 1 Useful Information (continued)
7- 1 Address Definitions
8- 1 Bit Definitions
11- 1 Alternate Entry Points and Pointer Table
13- 1 ..TEST 0 - J-11 ISP Test
24- 1 MEMORY TESTS
25- 1 ..TEST 1 - Program Memory (Sweep Bits)
26- 1 ..TEST 2 - Program Memory (Vector area)
27- 1 ..TEST 3 - Program Memory (Partition area)
28- 1 FAULT Lamp Routine
29- 1 ..TEST 4 - Rx33 Controller Hardware Test(s)
30- 1 ..TEST 4 PART II Interrupts & DMA
33- 1 Register & hardware test error traps
34- 1 ..TEST 6 - Find a good Rx33 drive.
37- 1 ..TEST 7 - Transfer Control to Loaded Image
40- 1 RXREST- perform a RECAL/VERIFY operation
42- 1 Read Boot Blocks to Memory
44- 1 RXREAD- read disk blocks to memory
47- 1 Rx33 Jack of All trades Subroutine
50- 1 Rx33 Local Interrupt Handler
51- 1 WAIT 33.2us Subroutine
52- 1 EXECOM - Common command interlock & execution routine
53- 1 Moving Inversions Test Description
54- 1 Moving Inversions Test

```
1 000000                        ident  03,01,( ROM Bootstrap )
                                .title MSC70 ROM Bootstrap  X03-01
                                .sbttl ***** MSC70 ROM Bootstrap version X03-01 *****
                                .ident /X03001/
 2                              ;+-------------------------------------------------------------+
 3                              ;  JJJJJJJJJJJJJJJ      11111  11111    RRRRRRRR      XX     XX
 4                              ;       JJJJ             111    111     RR     RR      XX   XX
 5                              ;       JJJJ             111    111     RR     RR       XX XX
 6                              ;       JJJJ     ::::    111    111     RR   RR          XXX
 7                              ;       JJJJ     ::::    111    111     RRRRRR          XX XX
 8                              ;       JJJJ             111    111     RR   RR        XX   XX
 9                              ;  JJ   JJJJ             1111   1111    RR    RR      XXX    XX
10                              ;  JJJJJJJJJ            111111 111111   RR     RR    XXX      XX
11                              ;
12                              ;  J11  ROM  BOOTSTRAP
13                              ;  (Rx33 FLOPPY DRIVE BASED)
14                              ;+-------------------------------------------------------------+
15                              ;program name:  MSC70 J11 ROM Bootstrap
16                              ;
17                              ;
18                              ;
19                              ;program abstract:
20                              ;       This program is designed to execute from the bootstrap ROM located
21                              ;       on the J-11 P.ioc module.  The bootstrap consists of tests of the J-11
22                              ;       CPU, part of the Program P.ioc memory, and the Rx33, followed by
23                              ;       loading of the next part of the bootstrap sequence from the Rx33.
24                              ;       The purpose of the bootstrap tests is to provide confidence that
25                              ;       the P.ioc can be used to load further software via the Rx33.
26                              ;       The software loaded as the last action of the bootstrap program
27                              ;       consists of more extensive tests of the P.ioc module.  After
28                              ;       these tests finish execution, the MSC70 Functional Software,  or
29                              ;       the offline software is loaded. Program control then transfers to
30                              ;       one of these packages.
31                              ;
32                              ;
33                              ;copyright statement:
34                              ;       Copyright 1985, Digital Equipment Corporation
35                              ;
36                              ;disclaimer:
37                              ;       The information in this document is subject to change without
38                              ;       notice and should not be construed as a commitment by Digital
39                              ;       Equipment Corporation.  Digital Equipment Corporation assumes
40                              ;       no responsibility for any errors that may appear in this doc-
41                              ;       ument.
42                              ;
43                              ;       The software described in this document is furnished under a
44                              ;       license and may only be used or copied in accordance with the
45                              ;       terms of such license.
46                              ;
47                              ;       Digital Equipment Corporation assumes no responsibility for the
48                              ;       use of reliability of its software on equipment that is not sup-
49                              ;       plied by Digital.
50                              ;+-------------------------------------------------------------+
```

```
                ;+-----------------------------------------------------------------------+
                ;
                ; AUTHOR - VERSION - DATE LIST
                ;
                ; Mike Moore    X01-00  AUG 1983  Original version with Tu58 load.
                ; Rod Li  A     X02-00  Feb 1985  Rx32 version.
                ; Rod Lilak     X02-01  Apr 1985  Rx33 version using DMA to load.
                ; Rod Lilak     X02-02  Apr 1985  Add BOOTAT, terminate command on error code.
                ; Rod Lilak     X02-03  Apr 1985  Added interrupt based read; this version only.
                ; Rod Lilak     X03-00  Oct 1985  Rewrite/optimize. DMA & interrupt tests added.
                ; Rod Lilak     X03-01  Nov 1985  Rev'd common execution timeout for rev. C J11.
                ;+-----------------------------------------------------------------------+
```

```
                        .sbttl  Useful Information
                        ;+-------------------------------------------------------------+
                        ;
                        ;  1) MAXIMUM PROGRAM SIZE - This program must fit into a 2,048 byte ROM.
                        ;             (Locations 17773000 thru 17776777)
                        ;
                        ;  2) Special Assembly Files Required - None. This is a stand-alone program.
                        ;
                        ;  3) Special use of MMU PARs (page address registers) - MMU  PARs
                        ;        are used as a convenient place to store information that
                        ;        must be passed along to the Init P.ioc test which succeeds the boot
                        ;        and are used as test locations. The contents of the USER PAR set are
                        ;        preserved.  This is where boot reasons and failing addresses are stored.
                        ;
                        ;     PAR usage summary:
                        ;             USER                KERNEL
                        ;     0       preserved           TEST 0
                        ;     1       preserved           initial stack
                        ;     2       preserved           initial stack
                        ;     3       preserved           initial stack
                        ;     4       preserved           initial stack/register bit tests
                        ;     5       preserved           unit # & type of boot (R3 on entry)
                        ;     6       preserved           CSR of Rx33 to use (R4 on entry)
                        ;     7       preserved           Validation of unit/type/CSR
                        ;
                        ;  4) Warm Boot entry restrictions: When using the 'WARM' boot feature,
                        ;        the ROM must be entered with interrupts disabled (PRI=7), with
                        ;        the MMU off, or in Kernel mode with the I/O page mapped.
                        ;
                        ;  5) Processor-time-dependent loops - All instruction sequences labeled 'PTIMEn'
                        ;        are timing loops that depend on the basic cycle time of the J-11 CPU chip.
                        ;
                        ;  6) ROM VERSION NUMBER - The Version and Edit numbers of the ROM software are
                        ;        stored in the last word of the ROM (address 176776).  The low byte is the
                        ;        version number, the high byte is the edit number.
                        ;
                        ;+-------------------------------------------------------------+
```

```
                        .sbttl  Useful Information (continued)
                        ;+-------------------------------------------------------------------------+
                        ;
                        ;       7) RX33 ERROR CODE TABLE - This program constructs an error table which
                        ;          remembers why the Rx33 subsystem could not be used for booting.
                        ;
                        ;            ADDRESS    |      MEANING
                        ;       ------------------------------------------------------
                        ;             400       | Contains controller error code (codes 1 & 2 ).
                        ;             402       | Rx33 address being accessed, if applicable.
                        ;             404       | Expected result.
                        ;             406       | Actual result.
                        ;             410       | Drive error code, byte-encoded : DRIVE_1/DRIVE_0
                        ;       ------------------------------------------------------
                        ;  400  CONTROLLER ERROR|      FAILURE INFORMATION
                        ;       ------------------------------------------------------
                        ;             1         | General. NXM occurred while accessing Rx33 registers.
                        ;             2         | A bit was stuck in the registers. See expected/actual.
                        ;             3         | Force mode interrupt did not occur.
                        ;             4         | DMA test mode hardware error.
                        ;             5         | DMA address counters wrong after transfer.
                        ;             6         | Incorrect data after DMA test operation.
                        ;             7         | Data parity bad after DMA test operation.
                        ;       ------------------------------------------------------
                        ;  410  DRIVE ERRORS    |      FAILURE INFORMATION
                        ;       ------------------------------------------------------
                        ;             10        | Device not ready- (no disk, or door open).
                        ;             11        | Hard error , CRC -or- DNF on recal/verify.
                        ;             12        | Track_0 bit not set after recal.
                        ;             13        | Seek command timeout.
                        ;             14        | Seek error (CRC -or- DNF)
                        ;             15        | Read sector command timeout.
                        ;             16        | Hard error (CRC -or- DNF) on read.
                        ;             17        | Non-bootable image.
                        ;       ------------------------------------------------------
                        ; In location 410 it is possible to have failure information for both drives.
                        ; If this is the case you will have non-zero data in both bytes. Only on the case
                        ; where failures are detected on both drives does the boot rom generate a
                        ; 'LOADFAL' failure code and branch to the fault light routine.
                        ;+-------------------------------------------------------------------------+
```

```
                ;+-----------------------------------------------------------------------
                ;process: This test is initiated each time the P.ioc subsystem is booted
                ;       or rebooted. Different entry points to the ROM allow the software
                ;       to determine whether the subsystem was being booted or rebooted.
                ;       Testing begins with the ISP of the J-11 processor.  All basic
                ;       PDP-11 instructions and all eight addressing modes are tested.
                ;       Following the J-11 ISP tests, 9 kwords of the P.ioc Program memory
                ;       are tested for proper addressing and ability to hold data and
                ;       proper parity.  The final test locates a working Rx33 drive
                ;       that contains an MSC70 formatted floppy and reads the first 8 blocks
                ;       from the diskette to the 8 kword partition in the Program memory.
                ;
                ;outputs:(to 'Init P.ioc Test')
                ;
                ;       case 1 - J-11 test passes, 8 kword partition ok, Rx33 ok
                ;              R0 = unit number of Rx33 used for boot (lower byte)
                ;                   type of boot (upper bit of upper byte) 0=cold, 1=warm
                ;              R1 = CSR address of Rx33 used for boot
                ;              R2 = Base address of tested 8 kword partition
                ;              'INIT' lamp is on
                ;
                ;       case 2 - J-11 test fails
                ;              No parameters passed.  J-11 'hangs' immediately (branch self)
                ;              'Init' lamp does not light, 'fault' lamp off
                ;
                ;       case 3 - No working 8 kword partition found in Program Memory
                ;               (or first 2048 bytes of Program Memory are faulty)
                ;              'Init' lamp is lighted
                ;              'Fault' lamp is lighted
                ;              Depressing fault switch causes blinking fault code display
                ;              'Fault code' = (memory test failure)
                ;              MSC70 must be rebooted to attempt recovery
                ;              Program Memory/Rx33 controller module probably needs replacing
                ;
                ;       case 4 - A failure was detected in the Rx33 controller logic test
                ;              'INIT' lamp is lighted
                ;              'Fault' lamp is lighted
                ;              Depressing fault switch causes blinking fault code display
                ;              'Fault code' = (memory test failure)
                ;              MSC70 must be rebooted to attempt recovery
                ;              Program Memory/Rx33 controller module probably needs replacing
                ;                    . 'or code table contains reason why each Rx33 unit
                ;              failed. See 'Rx33 Error Table' under USEFUL INFORMATION (above)
                ;
                ;       case 5 - No working Rx33 found with bootable media mounted
                ;              'Init  lamp is lighted
                ;              'Fault  lamp is lighted
                ;              Depressing fault switch causes blinking fault code display
                ;              'Fault code  = (Rx33 test failure)
                ;              MSC70 must be rebooted to attempt recovery
                ;              No Rx33 has bootable image mounted, or Rx33s are faulty
                ;              (Rx33 error code table contains reason why each Rx33 drive unit
                ;              failed. See  Rx33 Error Table  under USEFUL INFORMATION (above)
                ;
                ;+-----------------------------------------------------------------------
```

```
                        .sbttl  Address Definitions
                        ;+------------------------------------------------------------------------
                        ;
                        ;       Hardware Address Definitions
                        ;       For J11 CPU register
                        ;+------------------------------------------------------------------------
172340                          KUPA0= 172340           ;address of kernel PAR 0
172342                          KUPA1= 172342           ; kernel PAR 1
172344                          KUPA2= 172344           ; kernel PAR 2
172346                          KUPA3= 172346           ; kernel PAR 3
172350                          KUPA4= 172350           ; kernel PAR 4
172352                          KUPA5= 172352           ; kernel PAR 5
172354                          KUPA6= 172354           ; kernel PAR 6
172356                          KUPA7= 172356           ; kernel PAR 7

177572                          MMSR0= 177572           ; MMU STATUS REGISTER 0
177574                          MMSR1= 177574           ; MMU STATUS REGISTER 1
177576                          MMSR2= 177576           ; MMU STATUS REGISTER 2

170040                          RPIOCS= 170040          ; P.IOC CONTROL AND STATUS REGISTER
170042                          SWDCSR= 170042          ; SWITCH REGISTER CSR

                        ;+------------------------------------------------------------------------
                        ;       Logical Address Definitions
                        ;+------------------------------------------------------------------------

000400                          ERTADD= 400             ; Unique code for controller
000410                          ERTADG= 410             ; base of byte location for drive error table
173000                          basadd= 173000          ; first byte in ROM
176777                          lasadd= BASADD+3777     ; last byte in ROM
172340                          TESTNO= KUPAR0          ; Kernel PAR 0 is used to hold the test number
000230                          RXIVEC= 230             ; Rx33 interrupt vector
000232                          RXPSW = 232             ; Rx33 PSW on interrupt routine
000414                          INTFLG= 414             ; This will be the interrupt flag location
                        ;+------------------------------------------------------------------------
```

```
                        .sbttl  Bit Definitions
                        ;+------------------------------------------------------------------------
                        ;
                        ;       Bit definitions
                        ;
                        ;+------------------------------------------------------------------------

000001                  bit0    =       1
000002                  bit1    =       2
000004                  bit2    =       4
000010                  bit3    =       10
000020                  bit4    =       20
000040                  bit5    =       40
000100                  bit6    =       100
000200                  bit7    =       200
000400                  bit8    =       400
001000                  bit9    =       1000
002000                  bit10   =       2000
004000                  bit11   =       4000
010000                  bit12   =       10000
020000                  bit13   =       20000
040000                  bit14   =       40000
100000                  bit15   =       100000

040000                          PGMSIZ= 16384.  ;size of program memory partition to be tested in bytes
004000                          VECSIZ= 2048.   ;size of vector area to be tested in bytes
000010                          LODSIZ= 8.      ;# of blocks to load for boot

                        ;+
                        ;-------------------------------------------------------------------------
                        ;
                        ;       Fault register code definitions
                        ;
                        ;-------------------------------------------------------------------------
                        ;-

000021                  procfal=21      ; P.iop module failure
000022                  memfal=22       ; Memory module failure
000023                  loadfl=23       ; load device/subsystem failure code

                        ;+
                        ;-------------------------------------------------------------------------
                        ;
                        ;       P.iop Control and Status Register Bit Definitions
                        ;
                        ;-------------------------------------------------------------------------
                        ;-

100000                  TERMEN=100000   ;1 when terminal enable, 0 when Secure
004000                  SWPCMB=004000   ;1 when Swap  Control memory banks
002000                  SWPPMB=002000   ;1 when Swap Private memory banks
001000                  SELP1 =001000   ;Bit 1 of P.iop interrupt code
000400                  SELP0 =000400   ;Bit 0 of P.iop interrupt code
000200                  ENXARB=000200   ;1 when enable xmem arbitration
000100                  CLRBIT=000100   ;1 when clear I/O page device registers
000040                  HIPTST=000040   ;1 when enable high byte parity test
000020                  LOPTST=000020   ;1 when enable low byte parity test
```

```
                000010          LEDBIT=000010   ;1 when light STATE LED
                000004          NMABIT=000004   ;1 when create a non-memory-access cycle
                000002          CNTIE =000002   ;1 when control memory interrupts enabled
                000001          LCKBIT=000001   ;1 when create a control memory lock cycle

                                ;+
                                ;-------------------------------------------------------------
                                ;       Special local definition of PDP-11 RESET instruction
                                ;       (Required since RESET is illegal in normal MBC70 code,
                                ;       and MBC70 Macro Library defines RESET to cause an error)
                                ;
                                ;-------------------------------------------------------------
                                ;-
                000005          RESETS= 000005  ;define our own mnemonic for a RESET instruction
```

```
                        ;+
                        ;-------------------------------------------------------------
                        ;
                        ;       RX33 Register/Command Bit Definitions
                        ;
                        ;-------------------------------------------------------------
                        ;-


177400          RXCSR   =177400         ; RX32 CSR EQUATE
177402          RXTRK   =177402         ; CURRENT TRACK/LO MAR ADDRESS
177404          RXSEC   =177404         ; DESIRED SECTOR/NEXT MAR BYTE
177406          RXDAR   =177406         ; DESIRED TRACK/HIGH MAR ADDRESS
000017          MAXSEC  = 15.
000117          MAXCYL  = 79.           ;
004502          MAXLBN  = <MAXCYL*MAXSEC*2>
000017          PERsid  = maxsec
000036          PERcyl  = (PERsid*2)
                ; command definitions sorted by type
001000          MTRENA  = 1000          ; MOTOR ENABLE IS BIT9
021000          DENA    = 21000         ; MOTOR ENABLE + INTERRUPT ENABLE
000000          DRV0    = 0             ; DRIVE SELECT_0 IS , NECESSARILY, 0
000400          DRV1    = 400           ; BUT DRIVE SELECT 1 IS NOT
010000          DISRQ   = 10000         ; DISABLE REQUEST (NO DMA)
000004          VERIFY  = 4             ; VERIFY AFTER STEP/SEEK OPERATION (CHECKS HEADER)
000020          MULTI   = 20            ; MULTIPLE RECORD READ
002000          LED     = 2000          ; TURNS ON THE 'LED ON THE CONTROLLER BOARD
040000          DMATST  =  40000        ; DMA test mode bit
100000          MSTDON  = 100000        ; Bit 15 to turn on the module On led in RXCSR

                ;positioning commands
000010          RECAL   = 10            ; RESTORE TO TRACK 0
000020          SEEK    = 20            ; SEEK WITH HEADS UNLOADED AT START
000030          SEEKL   = 30            ; seek with heads loaded at start
000060          STEP    = 60            ; STEP WITH TRACK UPDATE (JUST UPDATES COUNT UNLESS VERIFY SET.
000120          STEPIN  = 120           ; STEP IN WITH TRACK UPDATE. DITTO THE ABOVE
000160          STEPOUT = 160           ; STEP OUT WITH TRACK UPDATE.DITTO THE DITTO ABOVE

                ;;interrupt mode commands.
000320          RSINT   = 320           ; reset interrupt modes
000330          FRCINT  = 330           ; FORCE UNCONDITIONAL INTERRUPT
000324          INTIDX  = 324           ; INTERRUPT ON EACH INDEX PULSE
000323          INTRDY  = 323           ; INTERRUPT ON DRIVE COMING READY AFTER OPERATION (not ready)

                ;data transfer commands .
000000          SIDE0   = 0
000002          SIDE1   = 2             ; VALUES TO UPDATE SIDE SELECT
000214          RDSEC   = 214           ; READ SECTOR BASIC COMMAND :REQUIRES MORE ARGUMENTS
000254          WRSEC   = 254           ; WRITE SECTOR BASIC COMMAND : ALSO REQUIRES EXTRA ARGUMENTS
000304          RDHDR   = 304           ; READ ADDRESS (HEADER)
000340          RDTRK   = 340           ; READ TRACK
000360          WRTRK   = 360           ; WRITE TRACK

                ; Frequently used command combinations.
001024          FDD =   MTRENA!SEEK!VERIFY      ; what you need to seek correctly. A LOCAL
021024          SRV =   DENA!SEEK!VERIFY        , SEEK+VERIFY+ MOTOR_ENABLE+INTERRUPT_ENABLE

```

```
                        ; States values.
100000          MEMPE   = 100000        ; memory parity error
040000          MEMNXM  = 40000         ; memory NXM
000010          CRCERR  = 10            ; crc error on data transfer
000020          RNFERR  = 20            ; record not found
000020          SEKERR  = 20            ; seek error for positioning commands
000200          MSKRDY  = 200           ; NOT READY when set
000004          MSKDLT  = 4             ; data late
000001          MSKBSY  = 1             ; busy bit
000004          TRACK0  = 4             ; track 0 status bit for seek/recal commands
                ;+--------------------------------------------------------------------+
```

```
                        ;+
                        ;-------------------------------------------------------------
                        ;
                        ;       P.ioc Switch/Display Register Bit Definitions
                        ;
                        ;-------------------------------------------------------------
                        ;-

004000                  FLTSW =004000   ;1 when 'Fault' switch depressed
002000                  BLKSW =002000   ;1 when 'Online' switch depressed
001000                  HBSW  =001000   ;1 when 'First Unmarked' switch depressed
000400                  HASW  =000400   ;1 when 'Second Unmarked' switch depressed
000020                  INILMP=000020   ;1 when light 'Init' lamp
000010                  FLTLMP=000010   ;1 when light 'Fault' lamp
000004                  BLKLMP=000004   ;1 when light 'Online' lamp
000002                  HBLMP =000002   ;1 when light 'First Unmarked' lamp
000001                  HALMP =000001   ;1 when light 'Second Unmarked' lamp
000037                  ALLLMP=HALMP!HBLMP!BLKLMP!FLTLMP!INILMP ;Inclusive OR of all lamp bits
                    ;+-------------------------------------------------------------
```

```
                        .sbttl  Alternate Entry Points and Pointer Table

                        ;+
                        ;*****************************************************************************
                        ;
                        ;       Alternate entry points to this program are supplied to allow
                        ;       the distinction between a 'cold' boot and a 'warm' boot.
                        ;       (A cold boot is the result of power-up, or  173000G' to ODT.  No
                        ;       saved context exists for a cold boot.  A warm boot indicates that
                        ;       there is saved context in the User PAR set, to be used by the Init
                        ;       Proc test.) The forced boot mechanism is preserved from the old boot
                        ;       ROM for compatibility.
                        ;
                        ;inputs:
                        ;               R3 = unit # of Rx33 to use for boot (lower byte)
                        ;               R4 = CSR address of Rx33 to use for boot (16 bits)
                        ;               R5 = Validation of unit/CSR (R3 XOR R4)
                        ;               For Warm boots, the User PAR set contains the reason
                        ;               for the reboot, and other context.  This context is not
                        ;               used by the Bootstrap ROM, it is merely not disturbed.
                        ;
                        ;process:       Set up a 'type of boot' parameter, depending upon which
                        ;               entry into the ROM is used.  The following entry points
                        ;               are provided:
                        ;                       1) Entry for 'Cold' boot
                        ;                       2) Entry for 'Warm' boot
                        ;
                        ;outputs:       R3, R4, and R5 preserved
                        ;               'Type of boot' encoded in upper  bit of R3
                        ;
                        ;               The value of the upper  bit of R3 is as follows:
                        ;                       cold boot = 0
                        ;                       warm boot = 1
                        ;
                        ;notes:
                        ;
                        ;History of changes to this routine
                        ;
                        ;*****************************************************************************
                        ;-
```

```
                        .Asect
                        .=basadd
                                ;+
                                ;-----------------------------------------------------------------
                                ;
                                ;       Entry for 'Cold' boot
                                ; This is starting address as defined by the hardware after power-up or INIT
                                ;-----------------------------------------------------------------
                                ;-

                        coldbt::

                                mtps    #340    ;prevents real time clock from interrupting
                                jmp     J11TST  ;cold boot entry


                                ;+
                                ;-----------------------------------------------------------------
                                ;
                                ;       Entry for 'Warm' boot   (base address + 10)
                                ;
                                ;-----------------------------------------------------------------
                                ;-

                        BTYPE2: bis     #100000,R3      ;set R3 negative to indicate warm boot
                                br      ALTENT          ;start bootstrap tests


                                ;+
                                ;-----------------------------------------------------------------
                                ;
                                ;       Table of pointers to ROM routines that are used from RAM
                                ;       (by the INIT P.IOC test).
                                ;
                                ;       The relative position of these pointers, with respect to the
                                ;       starting address of the ROM, must NEVER be changed.
                                ;
                                ;-----------------------------------------------------------------
                                ;-

                        Rx33SY: .word   RXREST  ;base + 16      Rx33 restore/synchronization routine
                        Rx33RD: .word   RXREAD  ;base + 20      Rx33 read routine
                        Rx33SN: .word   SENDPK  ;base + 22      Send Rx33 command packet (dummy vector now)
                        MTMENT: .word   MOVINV  ;base + 24      Moving inversions RAM test
                        ERRRPT: .word   ANYFAL  ;base + 26      Fault code display routine
                        Rx33RC: .word   RECVPK  ;base + 30      Receive Rx33 command or data packet (dv now)

                                ;****************** END OF POINTER TABLE **********************
```

```
                        .sbttl  ..TEST 0 - J-11 ISP Test
                        ;+
                        ;*****************************************************************************
                        ;       Test the basic PDP-11 instruction set and addressing modes.
                        ;
                        ;Inputs:        PSW Priority field = 7 (interrupts disabled) upon entry
                        ;               R3 = Unit # to use for boot (lower byte, value of 0 or 1 only)
                        ;               R4 = CSR address of Rx33 to use for boot (16 bit I/O page address)
                        ;               R5 = Validation for R3 and R4 (R3 XOR R4)
                        ;                    (validation prevents spurious device selection)
                        ;
                        ;Process:  Test the Instruction Set Processor (ISP) of the P.ioc's J-11
                        ;         processor.
                        ;       Instructions tested:
                        ;               CLR(B),COM(B),INC(B),DEC(B),NEG(B),TST(B),ROR(B),ROL(B),
                        ;               ASR(B),ASL(B),SWAB,NOP,ADC(B),CCC,CLN,CLV,CLZ,SCC,SEN,SEV,SEZ
                        ;
                        ;               MOV(B),CMP(B),BIT(B),BIC(B),BIS(B),ADD,SUB
                        ;
                        ;               BR,BNE,BEQ,BPL,BMI,BCC(BHIS),BCS(BLO),BGE
                        ;               BLT,BGT,BLE,BHI,BLOS,BVC,BVS
                        ;
                        ;               JMP,JSR,RTS,SOB,MTPS,MFPS
                        ;
                        ;       Addressing modes tested:
                        ;               All eight addressing modes are tested
                        ;
                        ;       Instructions NOT tested:
                        ;               ASH,ASHC,BPT,EMT,HALT
                        ;               IOT,MARK,MFPI,MTPI,MUL,RESET,RTI,RTT,SBC
                        ;               SPL,TRAP,WAIT,XOR
                        ;
                        ;       J-11 Logic NOT tested:
                        ;               Memory Mapping, interrupt priority and arbitration,
                        ;               Stack Overflow, Power Fail, Bus Time-out Logic
                        ;
                        ;       P.ioc logic NOT tested
                        ;               Control Memory Windows, ECC tables
                        ;               Operator console interface, 'K' status registers
                        ;               Control and arbitration of Data and Control buses
                        ;               Remainder of Program Memory (total size minus 8 Kwords)
                        ;
                        ;Outputs:
                        ;       Case 1 - no failure detected
                        ;               'Init' lamp is lighted
                        ;               Control passed to Program Memory Test
                        ;               R3,R4, and R5 are preserved in MMU Kernel PAR's 5,6, and 7
                        ;
                        ;       Case 2 - failure detected
                        ;               J-11 CPU is 'hung'.  PC of 'hung' identifies failure
                        ;               'Init' lamp is out, 'Fault' lamp is out
                        ;
                        ;Notes:
                        ;History of changes to this test:
                        ; 21-Oct-85 added mini divide test as part of CPU test
                        ;*****************************************************************************
                        ;-
```

```
                                ;+-------------------------------------------------------------
                                ;       Initialize stack pointer such that if a trap or interrupt
                                ;       occurs, the J-11 will generate a double bus error
                                ;
                                ;-------------------------------------------------------------

173032  042703  100000          J11TST: bic     #100000,R3      ;set 'type of boot' to 'cold boot'
173036  000005                  ALTENT: RESET                   ;clear all DCP lamps, clear all K interrupts
173040  012737  000414  177746          MOV     #414,@#177746   ;disable cache
173046  005067  177266                  clr     TESTNO          ;set test 0 to 0
173052  005006                          clr     SP              ;SP gets a 0
173054  005106                          com     SP              ;SP gets 177777, causes DBL BUS ERR if trap occurs

                                ;       MOV     #52,@#177564    ; * * * * temporary set up for 9600 baud console *

                                ;+
                                ;-------------------------------------------------------------
                                ;
                                ;       Test Branch and Condition Code Instructions
                                ;
                                ;       Test BR, BNE, BEQ, BPL, BMI, BCC(BHIS), BCS(BLO)
                                ;            BGE, BLT, BGT, BLE, BHI, BLOS, CCC, SCC
                                ;            SEN, SVS, SVC, SEN, CLN, SEZ, CLZ, SEC, CLC
                                ;
                                ;-------------------------------------------------------------
                                ;-

                                        scc                     ; N (- 1, Z (- 1, V (- 1, C (- 1
173056  000277                          bne     .               ;nbe (no branch expected)
173060  001377                          bpl     .               ;nbe
173062  100377                          bcc     .               ;nbe
173064  103377                          bgt     .               ;nbe
173066  003377                          bhi     .               ;nbe
173070  101377                          blt     .               ;nbe
173072  002777                          bvc     .               ;nbe
173074  102377                          cln                     ;n (- 0
173076  000250                          bge     .               ;nbe (n clear, v set)
173100  002377                          sen                     ;n (- 1
173102  000270                          blt     .               ;nbe (n set, v set)
173104  002777                          clc                     ;c (- 0
173106  000241                          bcs     .               ;nbe
173110  103777                          scc                     ;c (- 1
173112  000261                          bcc     .               ;nbe
173114  103377                          clz                     ;z (- 0
173116  000244                          beq     .               ;nbe
173120  001777                          sez                     ;z (- 1
173122  000264                          bne     .               ;nbe
173124  001377                          clv                     ;v (- 0
173126  000242                          bvs     .               ;nbe
173130  102777                          sev                     ;v (- 1
173132  000262                          bvc     .               ;nbe
173134  102377                          ccc                     ; N (- 0, Z (- 0, V (- 0, C (- 0
173136  000257                          beq     .               ;nbe (no branch expected)
173140  001777                          bmi     .               ;nbe
173142  100777                          bcs     .               ;nbe
173144  103777                          blt     .               ;nbe
173146  002777                          ble     .               ;nbe
173150  003777
```

'4'e

```
    58 173152  101777                    bles    .               ;nbe
    59 173154  102777                    bvs     .               ;nbe
    60 173156  000401                    br      10              ;branch forward
    61 173160  000777                    br      .               ;failed to branch forward
    62 173162  000270                10: sen                     ;set the N bit ( N (- 1 )
    63 173164  100377                    bpl     .               ;nbe (no branch expected)
    64 173166  002377                    bge     .               ;nbe
    65 173170  003377                    bgt     .               ;nbe
    66 173172  002402                    blt     20              ;should brar \
    67 173174  000777                    br      .               ;previous branch failed
    68 173176  003402                30: ble     100             ;should branch
    69 173200  100776                20: bmi     30              ;should branch backwards
    70 173202  000777                    br      .               ;backwards branch failed
    71
    72
    73 173204                        100: ;+
    74                                    ;-------------------------------------------------------------
    75                                    ;
    76                                    ;       Test CLR, COM, INC, DEC, NEG, TST, ROR, ROL, ASR, ASL, SWAB, NOP
    77                                    ;          and SEC
    78                                    ;
    79                                    ;       Address mode 0, using R0
    80                                    ;
    81                                    ;-------------------------------------------------------------
    82                                    ;-
    83
    84 173204  005000                    clr     R0              ; R0 (- 0,      C (- 0
    85 173206  005100                    com     R0              ; R0 (- 177777, C (- 1
    86 173210  005200                    inc     R0              ; R0 (- 000000, C = 1 (C not affected by inc)
    87 173212  005300                    dec     R0              ; R0 (- 177777, C = 1 (C not affected by dec)
    88 173214  [illegible]               nop                     ;insure nop does not interfere
    89 173216  103377                    bcc     .               ;hang here if c bit is clear, or bcc fails
    90 173220  005400                    neg     R0              ; R0 (- 000001, C (- 1
    91 173222  103377                    bcc     .               ;hang here if c bit is clear, or bcc fails
    92 173224  005700                    tst     R0              ; R0 = 000001,  C (- 0
    93 173226  001777                    beq     .               ;hang here - if R0 = 0, or tst or beq fails
    94 173230  006000                    ror     R0              ; R0 (- 000000, C (- 1
    95 173232  001377                    bne     .               ;hang here if R0 not 0
    96 173234  103377                    bcc     .               ;hang here if C didn't get loaded with a 1
    97 173236  006100                    rol     R0              ; R0 (- 000001, C (- 0
    98 173240  000240                    nop                     ;insure nop does not interfere
    99 173242  103777                    bcs     .               ;hang if c bit is set
   100 173244  001777                    beq     .               ;hang if R0 is 0
   101 173246  000261                    sec                     ; R0 = 000001, c (- 1
   102 173250  000300                    swab    R0              ; R0 (- 000400, C (- 0
   103 173252  103777                    bcs     .               ;hang if swab did not clear c bit
   104 173254  105700                    tstb    R0              ;test for lower byte of R0 = 0
   105 173256  001377                    bne     .               ;hang here if tstb failed, or R0 (lower) ne 0
   106 173260  000300                    swab    R0              ; R0 (- 000001, C (- 0
   107 173262  105700                    tstb    R0              ;test that lower byte of R0 now not zero
   108 173264  001777                    beq     .               ;hang if tstb failed, or R0 (lower) = 0
```

```
                                ;*
                                ;-------------------------------------------------------
                                ;
                                ;       Test MOV, CMP, BIT, BIC, BIS, ADD, SUB
                                ;
                                ;       Address mode 0, using R0, R1, and SP (R6)
                                ;
                                ;-------------------------------------------------------
                                ;-

173266  010600          mov     SP, R0          ; R0 <- 177777
173270  020600          cmp     SP,R0           ; R0 = 177777, SP = 177777
173272  001377          bne     .               ;hang if SP and R0 not equal
173274  040600          bic     SP,R0           ; R0 <- 000000
173276  001377          bne     .               ;hang if R0 not 0
173300  020600          cmp     SP,R0           ; R0 = 000000, SP = 177777
173302  001777          beq     .               ;hang if compare failed
173304  005200          inc     R0              ; R0 <- 000001
173306  060600          add     SP,R0           ; R0 <- 000000, C <- 1
173310  001377          bne     .               ;hang if add failed
173312  103377          bcc     .               ;hang if c bit did set
173314  050600          bis     SP,R0           ; R0 <- 177777
173316  005100          com     R0              ; R0 <- 000000
173320  001377          bne     .               ;hang if R0 not zero
173322  160600          sub     SP,R0           ; R0 <- 000001
173324  001777          beq     .               ;hang if R0 = 0
173326  005300          dec     R0              ; R0 <- 000000
173330  001377          bne     .               ;hang if R0 not 0
173332  005200          inc     R0              ; R0 <- 000001
173334  005001          clr     R1              ; R1 <- 000000
173336  005201          inc     R1              ; R1 <- 000001
173340  040001          bic     R0,R1           ; R1 <- 000000
173342  001377          bne     .               ;hang if R1 not 0
173344  005201          inc     R1              ; R1 <- 000001
173346  005100          com     R0              ; R0 <- 177776
173350  060100          add     R1,R0           ; R0 <- 177777,  C <- 0
173352  103777          bcs     .               ;will hang if C bit set
173354  005100          com     R0              ; R0 <- 000000
173356  001377          bne     .               ;hang if R0 not 0
```

```
                                        ;*
                                        ;-------------------------------------------------------------
                                        ;
                                        ;       Test CLRB, COMB, INCB, DECB, NEGB, TSTB, RORB, ROLB
                                        ;            ASRB, ASLB, MOVB, CMPB, BITB, BICB, BISB
                                        ;
                                        ;       Address mode 0, using R0 and R1
                                        ;
                                        ;-------------------------------------------------------------
                                        ;-

173360  005100          com     R0              ; R0 (- 177777
173362  110001          movb    R0,R1           ; R1 (- 177777 (movb to register sign extends)
173364  005101          com     R1              ; R1 (- 000000
173366  001377          bne     .               ;hang if R1 not 0
173370  110100          movb    R1,R0           ; R0 (- 000000 (movb to register sign extends
173372  001377          bne     .               ;hang if R0 not 0
173374  105100          comb    R0              ; R0 (- 000377
173376  105700          tstb    R0              ; R0 = 000377
173400  001777          beq     .               ;hang if lower byte is zero
173402  000300          swab    R0              ; R0 (- 177400
173404  105700          tstb    R0              ; R0 = 177400 (lower byte = 0)
173406  001377          bne     .               ;hang if lower byte of R0 not 0
173410  005101          com     R1              ; R1 (- 177777
173412  105001          clrb    R1              ; R1 (- 177400
173414  105701          tstb    R1              ; R1 = 177400
173416  001377          bne     .               ;hang if lower byte of R1 not 0
173420  000301          swab    R1              ; R1 (- 000377
173422  105701          tstb    R1              ; R1 = 000377
173424  001777          beq     .               ;hang if lower byte of R1 = 0
173426  120001          cmpb    R0,R1           ; R0 = 177400, R1 = 000377
173430  001777          beq     .               ;hang if lower bytes are equal
173432  000300          swab    R0              ; R0 (- 000377
173434  160300          sub     R0              ; R0 (- 000376, C (- 1
173436  103377          bcc     .               ;hang if c bit is not set
173440  120001          cmpb    R0,R1           ; R0 (lower) = 376, R1 (lower) = 377
173442  001777          beq     .               ;hang if lower bytes are equal
173444  105200          incb    R0              ; R0 (- 000377
173446  006300          asl     R0              ; R0 (- 000776
173450  105200          incb    R0              ; R0 (- 000777
173452  120001          cmpb    R0,R1           ; R0 = 000777, R1 = 000377 (equal in lower byte)
173454  001377          bne     .               ;hang if lower bytes are not equal
173456  105300          decb    R0              ; R0 (- 000776
173460  106200          asrb    R0              ; R0 (- 000777 (lower byte sign extends)
173462  120001          cmpb    R0,R1           ; R0 = 000777, R1 = 000377 (equal in lower byte)
173464  001377          bne     .               ;hang if lower bytes are not equal
173466  140100          bicb    R1,R0           ; R0 (- 000400 (lower byte cleared)
173470  001377          bne     .               ;hang if lower byte did not result in a 0
173472  150100          bisb    R1,R0           ; R0 (- 000777
173474  105100          comb    R0              ; R0 (- 000400 (lower byte 0)
173476  001377          bne     .               ;hang if lower byte not 0
173500  130100          bitb    R1,R0           ; R0 = 000400, R1 = 000377
173502  001377          bne     .               ;hang if bit test result was a 1
173504  105200          incb    R0              ; R0 (- 000401
173506  105101          comb    R1              ; R1 (- 000000
173510  105201          incb    R1              ; R1 (- 000001
173512  130100          bitb    R1,R0           ; R0 = 000401, R1 = 000001
```

```
 58 173514  001777          beq     .               ;hang if bit test result was zero
 59 173516  000241          clc                     ; C (- 0
 60 173520  106001          rorb    R1              ; R1 (- 000000, C (- 1
 61 173522  103377          bcc     .               ;hang if c bit is not a 1
 62 173524  001377          bne     .               ;hang if result in R1 not 0
 63 173526  106101          rolb    R1              ; R1 (- 000001, C (- 0
 64 173530  103777          bcs     .               ;hang if c bit is set
 65 173532  001777          beq     .               ;hang if result in R1 is 0
 66 173534  105401          negb    R1              ; R1 (- 000377, c (- 1
 67 173536  105201          incb    R1              ; R1 (- 000000, C remains 1 (not affected)
 68 173540  103377          bcc     .               ;hang if c bit clear
 69 173542  001377          bne     .               ;hang if result in R1 is not zero
 70 173544  000300          swab    R0              ; R0 (- 000401
 71 173546  105300          decb    R0              ; R0 (- 000400
 72 173550  001377          bne     .               ;hang if result in R0 (lower) is not 0
 73 173552  005700          tst     R0              ; R0 = 000400
 74 173554  001777          beq     .               ;hang if result in R0 is 0
```

```
                                        ;+
                                        ;-------------------------------------------------------------
                                        ;
                                        ;       Test JMP and SOB instructions
                                        ;
                                        ;       Using R0, R1, and R2 (jmp uses mode 6 PC addressing)
                                        ;
                                        ;-------------------------------------------------------------
                                        ;-

173556  005000                          clr     R0              ; R0 (- 000000
173560  005001                          clr     R1              ; R1 (- 000000
173562  005002                          clr     R2              ; R2 (- 000000
173564  005200                          inc     R0              ; R0 (- 000001
173566  005201                          inc     R1              ; R1 (- 000001
173570  006300                          asl     R0              ; R0 (- 000002
173572  006301                          asl     R1              ; R1 (- 000002
173574  105301                  sobbak: decb    R1              ; R1 (- R1 minus 1
173576  105202                          incb    R2              ; R2 (- R2 plus 1
173600  077003                          sob     R0,sobbak       ; R0 (- R0 minus 1, branch if result not 0
173602  105701                          tstb    R1              ; R1 should now be zero also
173604  001377                          bne     .               ;hang if R1 is not 0
173606  105302                          decb    R2              ; R2 (- 1
173610  105302                          decb    R2              ; R2 (- 0
173612  001377                          bne     .               ;hang if R2 did not equal 2 after loop above
173614  000167  000002                  jmp     jmptar          ;test a forward jump
173620  000777                          br      .               ;hang if jump failed
173622  000167  000010          jmptar: jmp     jmptr1          ;test another forward jump
173626  000777                          br      .               ;hang if jump failed
173630  000167  000010          jmptr2: jmp     J11A00          ;test another forward jump
173634  000777                          br      .               ;hang if jump failed
173636  000167  177766          jmptr1: jmp     jmptr2          ;test a backward jump
173642  000777                          br      .               ;hang if jump failed
```

```
                                ;+
                                ;-------------------------------------------------------------
                                ;
                                ;       Test addressing modes
                                ;
                                ;       Address modes 1, 2, 3, 4, 5, 6, and 7 are tested
                                ;
                                ;       R0 and R1 are used
                                ;
                                ;-------------------------------------------------------------
                                ;-

173644  012710  173732  J11AD0: mov     #maddat,R0      ;R0 gets pointer to data for this test  (SM2)
173650  011001                  mov     (R0),R1         ;R1 gets test data                      (SM1)
173652  022001                  cmp     (R0)+,R1        ;check data against itself              (SM2)
173654  001377                  bne     .               ;hang if datum not equal
173656  020027  173734          cmp     R0,#MDDAT+2     ;check R0 for correct contents
173662  001377                  bne     .               ;hang if R0 is wrong
173664  063001                  add     @(R0)+,R1       ;add test data to R1                    (SM3)
173666  020027  173736          cmp     R0,#MDDAT+4     ;check R0 for correct value
173672  001377                  bne     .               ;hang if wrong value in R0
173674  163001                  sub     @-(R0),R1       ;subtract test data from R1             (SM5)
173676  041001                  bic     -(R0),R1        ;clear R1 by bit clear with same data   (SM4)
173700  001377                  bne     .               ;hang if R1 was not cleared
173702  020027  173732          cmp     R0,#MDDAT       ;check R0 for correct value
173706  001377                  bne     .               ;hang if R0 is incorrect
173710  056001  000004          bis     4(R0),R1        ; R1 (- 177777                          (SM6)
173714  037001  000006          bit     @6(R0),R1       ;test for any bits set                  (SM7)
173720  001777                  beq     .               ;hang if no bits are set
173722  005101                  com     R1              ; R1 (- 000000
173724  001377                  bne     .               ;hang if R1 was not all ones before com
173726  000405                  br      J11MTP          ;transfer to next test
173730  000777                  br      .               ;previous branch failed

173732  173732          maddat: .word   maddat          ;data locations for addressing mode test
173734  173732                  .word   maddat
173736  177777          maddt1: .word   177777
173740  173736                  .word   maddt1
```

```
                                        ;+
                                        ;-------------------------------------------------------------
                                        ;
                                        ;       Test MFPS Instruction
                                        ;
                                        ;       Address mode 0
                                        ;
                                        ;       Note: Until interrupt logic is tested (in 'Init P.ioc test'
                                        ;             following the ROM test), testing of the priority bits
                                        ;             in the PSW are delayed to avoid uncontrolled detection
                                        ;             of failures in the interrupt logic.  This test restricts
                                        ;             itself to checking that the priority is now 7, and that
                                        ;             the N, Z, V, and C bits are set and cleared.
                                        ;
                                        ;-------------------------------------------------------------
                                        ;-

173742  000277          J11MTP: scc                     ; N (- 1, Z (- 1, V (- 1, C (- 1
173744  106700                  mfps    R0              ; R0 (- 177757 (lower byte of PSW, sign extend)
173746  020027  177757          cmp     R0,#177757      ; check for pri 7, all cc bits set
173752  001377                  bne     .               ;hang if mfps failed
173754  000257                  ccc                     ; N (- 0, Z (- 0, V (- 0, C (- 0
173756  106700                  mfps    R0              ; R0 (- 177740
173760  020027  177740          cmp     R0,#177740      ; check for pri 7, all cc bits clear
173764  001377                  bne     .               ;hang if mfps failed
                                        ;+
                                        ;-------------------------------------------------------------
                                        ;
                                        ;       Test JSR and RTS instructions
                                        ;       TURN OFF THE MMU
                                        ;
                                        ;       NOTE: In order to test JSR, the stack must be used.  This
                                        ;       requires using some location in memory for a one word stack.
                                        ;
                                        ;       This test uses part of the MMU's Kernel Page Address Register Set as
                                        ;       a stack area (8 contiguous 16 bit registers).
                                        ;
                                        ;-------------------------------------------------------------
                                        ;-
173766  005037  177572          clr     @#SR0           ;TURN OFF THE MMU
173772  012706  172352          mov     #KPAR5,SP       ;point SP to Kernel PAR5
                                                        ;(KPAR's 4,3,2,1, and 0 used for stack)
173776  004767  000006          jsr     PC,JSRTST       ;test JSR instruction. Return at call+6
174002  000777                  br      .               ;hang if JSR returns to wrong location
174004  000405                  br      J11ADC          ;Correct return from JSR above
174006  000777                  br      .               ;hang if JSR returned to wrong location

174010  062716  000002  JSRTST: add     #2,(SP)         ;update return address by 2
174014  000207                  rts     pc              ;test the RTS instruction
174016  000777                  br      .               ;hang if RTS failed to return
```

```
                                        ;+
                                        ;-------------------------------------------------------------
                                        ;
                                        ;       Test ADC and ADCB instructions
                                        ;
                                        ;       Move the boot parameters from R3, R4, and R5 to the Memory
                                        ;       Management Kernel PAR's 5, 6, and 7, where they will be
                                        ;       preserved across the memory and Rx33 tests to follow.
                                        ;
                                        ;-------------------------------------------------------------
                                        ;-


                                J11ADC:
174020  012701  172352                  mov     @#KPAR5,R1      ;point R1 to Kernel par 5
174024  010321                          mov     R3,(R1)+        ;save unit and type of boot in UPAR5
174026  010421                          mov     R4,(R1)+        ;save CSR of Rx33 in UPAR6
174030  010511                          mov     R5,(R1)         ;save validation word for unit and CSR
174032  005001                          clr     R1              ; R1 = 0, C = 0
174034  005501                          adc     R1              ; R1 = 0        (0+0=0)
174036  001130                          bne     CPUFAL          ;br if answer is non-zero
174040  103527                          bcs     CPUFAL          ;br if c bit is set
174042  105501                          adcb    R1              ; R1 = 0        (0+0=0)
174044  001125                          bne     CPUFAL          ;br if answer is non-zero
174046  005101                          com     R1              ; R1 = 177777, C = 1
174050  005501                          adc     R1              ; R1 = 0, C = 1 (177777+1=0)
174052  001122                          bne     CPUFAL          ;br if answer is not zero
174054  103121                          bcc     CPUFAL          ;br if c bit did not set
174056  012702  177777                  mov     @#177777,R2     ; R2 = 177777, C = 1
174062  105502                          adcb    R2              ; R2 = 177400, C = 1
174064  103115                          bcc     CPUFAL          ;br if c bit did not set
174066  020227  177400                  cmp     R2,@#177400     ;check for correct answer in R2
174072  001112                          bne     CPUFAL          ;br if wrong answer
174074  000261                          sec                     ; C = 1
174076  005501                          adc     R1              ; R1 = 1, C = 0 (0+1=1)
174100  103507                          bcs     CPUFAL          ;br if c bit is set
174102  005301                          dec     R1              ; R1 = 0        (1-1=0)
174104  001105                          bne     CPUFAL          ;br if R1 did not contain a 1
```

```
                                        ;+
                                        ;-------------------------------------------------------
                                        ;
                                        ;       Perform bit tests of R1 thru R5
                                        ;
                                        ;-------------------------------------------------------
                                        ;-

 174106  012701  100000                 mov     #100000,R1      ;preset all registers to 100000
 174112  010102                         mov     R1,R2
 174114  010203                         mov     R2,R3
 174116  010304                         mov     R3,R4
 174120  010405                         mov     R4,R5
 174122  010506                         mov     R5,SP
 174124  001475                         beq     CPUFAL          ;abort if bit was dropped along the way
 174126  010667  176216                 mov     SP,@#KPAR4      ;set Kernel PAR4 to same pattern
 174132  012700  000017                 mov     #15.,R0         ;R0 will count 15 shifts
 174136  006201                 100:    asr     R1              ;shift each register 1 bit right, sign extending is
 174140  006202                         asr     R2
 174142  006203                         asr     R3
 174144  006204                         asr     R4
 174146  006205                         asr     R5
 174150  006206                         asr     SP
 174152  006267  176172                 asr     @#KPAR4
 174156  020102                         cmp     R1,R2           ;check for new pattern, with one additional '1' bit
 174160  001057                         bne     CPUFAL
 174162  020304                         cmp     R3,R4
 174164  001055                         bne     CPUFAL
 174166  020506                         cmp     R5,SP
 174170  001053                         bne     CPUFAL
 174172  020667  176152                 cmp     SP,@#KPAR4
 174176  001050                         bne     CPUFAL
 174200  077022                         sob     R0,100          ;branch until 15 shifts have been performed
 174202  026727  176142  177777         cmp     @#KPAR4,#177777 ;is final pattern all 1s?
 174210  001043                         bne     CPUFAL
```

```
                                        ;+
                                        ;-------------------------------------------------------------
                                        ;
                                        ;       Basic Adder Test
                                        ;
                                        ;-------------------------------------------------------------
                                        ;-

 9 174212  012706  172352              MOV     #KPAR5,SP       ;set stack pointer to MMU Kernel PAR5
10 174216  005001                      CLR     R1              ; R1 = 0
11 174220  062701  125252              ADD     #125252,R1      ; R1 = 125252, N=1, Z=0, V=0  C=0
12 174224  062701  052525              ADD     #052525,R1      ; R1 = 177777, N=1, Z=0, V=0, C=0
13 174230  103433                      BCS     CPUFAL          ;br if c bit is set
14 174232  003032                      BGT     CPUFAL          ;br if n bit is not set
15 174234  102431                      BVS     CPUFAL          ;br if v bit is set
16 174236  062701  146314              ADD     #146314,R1      ; R1 = 146313, N=1, Z=0, V=0, C=1
17 174242  103026                      BCC     CPUFAL          ;br if c bit is not set
18 174244  062701  125252              ADD     #125252,R1      ; R1 = 073565, N=0, Z=0, V=1, C=1
19 174250  102023                      BVC     CPUFAL          ;br if v bit did not set
20 174252  103022                      BCC     CPUFAL          ;br if c bit did not set
21 174254  062701  104213              ADD     #-73565,R1      ; R1 = 000000, N=0, Z=1, V=0, C=1
22 174260  001017                      BNE     CPUFAL          ;br if R1 not zero
23 174262  102416                      BVS     CPUFAL          ;br if v bit is set
24 174264  103015                      BCC     CPUFAL          ;br if c bit is clear, else do divide test
25
26                                      ;+-------------------------------------------------------------
27                                      ; Basic divide test
28                                      ;
29                                      ;+-------------------------------------------------------------
30
31 174266  012702  177777      DIVTST: MOV     #-1,R2
32 174272  010203                      MOV     R2,R3                   ; R2, R3 both have minus one
33 174274  071227  177777              DIV     #-1,R2                  ; perform the divide
34
35 174300  100407                      BMI     CPUFAL                  ; check the flag state now
36 174302  103406                      BCS     CPUFAL
37 174304  102405                      BVS     CPUFAL
38
39 174306  005703                      TST     R3                      ; should be no remainder
40 174310  001003                      BNE     CPUFAL
41
42 174312  020227  000001              CMP     R2,#1                   ; resulting quotient
43 174316  001403                      BEQ     J11DON                  ; if not what we got
```

```
                                ;+
                                ;-------------------------------------------------------------
                                ;
                                ;       J-11 CPU test failure (only used for last few J-11 tests)
                                ;
                                ;       1) Set to R0 to fault code for "P.ioc failure"
                                ;
                                ;       2) Light the FAULT lamp
                                ;
                                ;       3) Monitor the FAULT switch
                                ;
                                ;-------------------------------------------------------------
                                ;-

174320                  CPUFAL:
174320  012700  000021                mov     #PIOFAL,R0      ;R0 gets error code for P.ioc failure
174324  000543                        br      ##WFAL          ;go to common error display routine

174326                  J11BGN: ;+
                                ;-------------------------------------------------------------
                                ;
                                ;       Light the 'init' lamp to show J-11 test passed
                                ;
                                ;-------------------------------------------------------------
                                ;-

174326  052737  000020  170042        bis     #INILMP,@#MEMSR ;light the init lamp to show that J tests passed
```

```
                        .sbttl  MEMORY TESTS
                        ;+
                        ;******************************************************************************
                        ;
                        ;       Test the first 2,048 bytes of Program Memory, and
                        ;       Find a working 8 Kword partition in Program Memory
                        ;
                        ;inputs:        MMU Kernel PAR 5 = unit # (lower byte), type of boot (upper byte)
                        ;               MMU Kernel PAR 6 = CSR of Rx33 to use for boot
                        ;               MMU Kernel PAR 7 = validation word for unit and CSR parameters
                        ;
                        ;process: Find an 8 Kword partition in Program Memory that passes a combined
                        ;       data integrity and addressing test.  In addition to this 8 Kword
                        ;       partition, the first 2048 bytes of Program memory must be working for
                        ;       use as the PDP-11 interrupt and trap vector area.
                        ;       If the first 2048 bytes do not work, the entire test
                        ;       is considered to have failed.  If the first 2048 bytes are working,
                        ;       then try to find any 8 Kword chunk of Program memory that works.  If the
                        ;       first 2048 bytes are working, but no contiguous 8 Kword partition can
                        ;       be found, the test fails.
                        ;
                        ;outputs:  case 1 - no failure detected
                        ;               'Init' lamp is on, 'Fault' lamp is off
                        ;               R0 = Base address of tested 8 Kword partition
                        ;               MMU Kernel PAR's 5,6, and 7 are unchanged
                        ;               Control is passed to the Rx33 test
                        ;
                        ;          case 2 - failure detected (see above for definition of failure)
                        ;               'Fault' lamp is on
                        ;               'Init' lamp is on
                        ;               J-11 is in loop, monitoring the fault lamp
                        ;               Depressing the fault lamp causes blinking display of fault code.
                        ;               J-11 must be rebooted to attempt recovery
                        ;
                        ;notes:
                        ;       1) the variable 'PARSIZ' defines the size of the partition to be tested
                        ;       2) the variable 'VECSIZ' defines the size of the vector area to test
                        ;
                        ;history of changes to t" : test:
                        ;
                        ;******************************************************************************
                        ;-
```

```
                        .sbttl  ..TEST 1 - Program Memory (Swap Bits)
                        ;+
                        ;*************************************************************************
                        ;
                        ;       Test the 'Swap Private/Control Memory Banks' bits in the P.ioc CSR
                        ;
                        ;inputs:  none
                        ;
                        ;process: 1) set swap bits
                        ;         2) Check that 'swap PMB' is set
                        ;         3) Clear the 'swap CMB' bit
                        ;         4) Check that the 'swap PMB' bit is still set
                        ;         5) Clear the 'swap PMB' bit
                        ;         6) Check that both swap bits are clear
                        ;
                        ;outputs: Declare a P.ioc failure if any of the tests fail
                        ;
                        ;*************************************************************************
                        ;-
174334  005267  176000          inc     TESTNO          ;set test number to 1
174340  012704  170040          mov     #DPIOCSR,R4     ;point R4 to P.ioc's CSR
174344  052714  006000          bis     #DSPCMB!DSPPMB,(R4) ;set both swap bits
174350  032714  004000          bit     #DSPCMB,(R4)    ;check the swap control mem bit
174354  001003                  bne     5$              ;br if bit is set
174356  042714  006000  4$:     bic     #DSPCMB!DSPPMB,(R4) ;try to clear both bits
174362  000756                  br      CPUFAL          ;declare a P.ioc failure
                        
174364  042714  004000  5$:     bic     #DSPCMB,(R4)    ;clear the swap control mem bit
174370  032714  006000          bit     #DSPCMB!DSPPMB,(R4) ;test both bits (one should be set)
174374  001770                  beq     4$              ;br if neither bit is set
174376  042714  002000          bic     #DSPPMB,(R4)    ;clear the swap private bit (both clear now)
174402  032714  006000          bit     #DSPCMB!DSPPMB,(R4) ;check for both bits clear
174406  001363                  bne     4$              ;br if either bit failed to clear
```

```
                                        .sbttl  ..TEST 2 - Program Memory (Vector area)
174410  005267  175724                  inc     TESTNB          ;set test number to 2
174414                          MEMTST: ;+
                                        ;-------------------------------------------------------------
                                        ;
                                        ;       Test the first 2048 bytes of program memory
                                        ;
                                        ;       If a failure occurs in the first 2048 bytes, then switch
                                        ;       the memory bank that occupies that address range, if
                                        ;       possible, and repeat test.  If the memory bank can not be
                                        ;       switched, or if the second bank also fails, declare error.
                                        ;
                                        ;-------------------------------------------------------------
                                        ;-
174414  005004                          clr     R4              ;address 0 is base address of area to test
174416  012705  003776                  mov     #VECSIZ-2,R5    ;R5 gets last address to test
174422  005003                          clr     R3              ;R3 gets 0 of failing addresses to store (none)
174424  004767  002226                  jsr     PC,MSUTST       ;call the memory test
174430  103012                          bcc     PARTST          ;br if memory test passed
174432                          BMTTST: ;+
                                        ;-------------------------------------------------------------
                                        ;
                                        ;       if( 'bank swap'=0 )  then ( 'bank swap' gets a 1, retry test )
                                        ;       if( 'bank swap'=1 ) then ( declare unrecoverable error )
                                        ;
                                        ;NOTE ON PROGRESSION OF 'SWAP BANK' SIGNAL:
                                        ;
                                        ;       Pass 1 - 'swap bank' zero
                                        ;       Pass 2 - 'swap bank' = one
                                        ; note - now there is only one 'swap' possible (private memory banks
                                        ;-------------------------------------------------------------
                                        ;-
174432  012706  172352                  mov     #KISAR5,SP      ;reset SP
174436  012703  170040                  mov     #KIPICS,R3      ;R3 points to P.iec Control and Status register
                                ;--   If ---->
174442  032713  002000                  bit     #SWAPMB,(R3)    ;are banks swapped now?
                                ;-- Then ---->
174446  001070                          bne     BADMEM          ;br if banks are now swapped
                                ;-- Else ---->
174450  052713  002000                  bis     #SWAPMB,(R3)    ;assert 'swap banks' signal
174454  000757                          br      MEMTST          ;retry memory test on first 2048 bytes
```

```
                                        .sbttl  ..TEST 3 - Program Memory (Partition area)
174456                          PARTST: ;+
                                        ;-------------------------------------------------------------------
                                        ;
                                        ;       Find a working 8 kword partition in Program Memory
                                        ;
                                        ;       If no failures are detected, then continue with Rx33 test.
                                        ;       If a failure is detected, set the base address of the area
                                        ;       to be tested to the next address following the failure, and
                                        ;       try to find a contiguous 8 kword area with no errors.  If no
                                        ;       8 kword area can be found, declare a memory test error.
                                        ;
                                        ;-------------------------------------------------------------------
                                        ;-
174456  005267  175536                  inc     TESTNO          ;set test # to 3
174462  012706  004000                  mov     #VECSIZ,SP      ;set stack pointer to top of tested 2048 bytes
174466  012737  176576  000004          mov     #ODDRTI,#04     ; in case you get nxm when writing CSR, don't fail
                                ; Have start the Rx33 spinning while test memory. (May get NXM.)
174474  012767  011014  002676          MOV     #INTRENA!DISRO!RECAL!VERIFY,@#CSR
                                ; now reset the NXM vector to do a real fail during memory test
174502  012737  174550  000004          mov     #MEMERR,#04     ;set timeout trap vector
174510  012767  000340  003270          mov     #340,6          ;set trap 4 psw word
174516  012737  174550  000114          mov     #MEMERR,#114    ;set parity trap vector
174524  012737  000340  000116          mov     #340,#116
174532  012704  004000                  mov     #VECSIZ,R4      ;R4 gets first address to test
174536  012705  043776                  mov     #VECSIZ+PARSIZ-2,R5 ;R5 gets last address to test
174542  004767  002110          MEMRTY: jsr     PC,MOVINV       ;perform the Moving-Inversions test
174546  103023                          bcc     MEMOKN          ;br if memory test passed
                                        ;+
                                        ;-------------------------------------------------------------------
                                        ;
                                        ;       R0 contains the memory address that failed
                                        ;
                                        ;       Add 2 to R0, then adjust R0 upwards (if necessary) to the
                                        ;       next even 32 word boundary. Start the test over there.
                                        ;
                                        ;       if (last address of next partition is beyond the end of memory)
                                        ;       then ( declare a program memory failure)
                                        ;       else ( test the newly defined partition)
                                        ;
                                        ;-------------------------------------------------------------------
                                        ;-
174550  012706  004000          MEMERR: mov     #VECSIZ,SP      ;reset SP in case we trapped to here
174554  010004                          mov     R0,R4           ;move failing address to R4
174556  062704  000002                  add     #2,R4           ;adjust address past failure
174562  032704  000077                  bit     #77,R4          ;is address on 32 word boundary?
174566  001404                          beq     10$             ;br if it is
174570  042704  000077                  bic     #77,R4          ;adjust address to 32 word boundary
174574  062704  000100                  add     #100,R4
174600  010405          10$:            mov     R4,R5           ;set R5 to starting address
174602  062705  037776                  add     #PARSIZ-2,R5    ;R5 now = last address of 8 kword partition to test
174606  020527  160000                  cmp     R5,#160000      ;see if we are into I/O page yet
174612  103006                          bhis    BADMEM          ;br if we are into I/O page
174614  000752                          br      MEMRTY          ;retry memory test on new partition

174616  010400          MEMOKN: mov     R4,R0                   ;R0 gets base address of 8 kword partition
174620  010006                  MOV     R0,SP                   ; stack pointer now at base of tested mem
```

```
    58 174622  062706  040000              ADD     #PMSIZ,SP               ; stack now at TOP of tested area
    59 174626  000441                      BR      R1REB                   ; now start first R133 test
    60                              ;+-------------------------------------------------------------------+
    61
```

```
                        .sbttl  FAULT Lamp Routine
                        .enable LSB
174630                  BADMEM: ;+
                                ;-------------------------------------------------------------
                                ;
                                ;       Declare a memory test error
                                ;       Set the fault code register (R0) to 'Memory Module Failure'
                                ;
                                ;-------------------------------------------------------------
                                ;-
174630  012700  000022          mov     #MEMFAL,R0      ;R0 gets code for 'memory module failure'
174634                  ANYFAL: ;+
                                ;-------------------------------------------------------------
                                ;       Handle Fault lamp report for any error. R0 contains failure code
                                ;
                                ;       Clear all lamps except INIT (indicates whether J-11 tests passed).
                                ;       Light the Fault lamp, then monitor the fault switch. When the Fault
                                ;       switch is pressed, display a blinking failure code in the lamps.
                                ;       This routine is stuck in the middle of the ROM so all can
                                ;       get at it with a branch instruction
                                ;-------------------------------------------------------------
                                ;-
174634  106427  000340          mtps    #340            ;disable interrupts
174640  012701  170042          mov     #MSCSR,R1       ;point R1 to P.ioc display control register
174644  042711  000017          bic     #FLTLMP!BLKLMP!HBLMP!HALMP,(R1) ;extinguish all lamps except INIT
174650  052711  000010          bis     #FLTLMP,(R1)    ;light the fault lamp
174654  005067  002520          CLR     RXCSR           ; spin down the floppies. (If you NXM here, bite it.)
174660  032711  004000  400:     bit     #FLTSW,(R1)     ;has the fault switch been depressed?
174664  001775                  beq     400             ;br if not
                                ;+
                                ;-------------------------------------------------------------
                                ;
                                ;Display the fault code in the lamps for 1/2 second
                                ;
                                ;-------------------------------------------------------------
                                ;-
174666  042711  000037  500:    bic     #ALLLMP,(R1)    ;extinguish all lamps
174672  010011                  mov     R0,(R1)         ;put fault code in lamps
174674  012703  000007  PTIME0: mov     #7.,R3          ;repeat 50 msec delay below 10 times
174700  012702  045570  600:    mov     #19320.,R2      ;R2 gets constant for 50 msec delay
174704  077201          700:    sob     R2,700          ;delay 50 msec
174706  077304                  sob     R3,600          ;delay 1/2 second (500 msec) total
                                ;+
                                ;-------------------------------------------------------------
                                ;
                                ;Extinguish all the lamps for 1/2 second
                                ;
                                ;-------------------------------------------------------------
                                ;-
174710  042711  000037          bic     #ALLLMP,(R1)    ;turn off all lamps
174714  012703  000005  PTIME1: mov     #5.,R3          ;repeat 50 msec delay below 10 times
174720  012702  045570  800:    mov     #19320.,R2      ;R2 gets constant for 50 msec delay
174724  077201          900:    sob     R2,900          ;delay 50 msec
174726  077304                  sob     R3,800          ;delay 1/2 sec (500 msec) total
174730  000756                  br      500             ;turn on the lamps for 1/2 second again
                        .dsabl LSB
                        ;+-------------------------------------------------------------------->
```

```
;58
;59
```

```
                        .sbttl  ..TEST 4 - Rx33 Controller Hardware Test(s)
                        ;+-----------------------------------------------------------------------
                        ;                       ROUTINE DESCRIPTION :
                        ;       This is the first test for the Rx33 subsystem.
                        ; First it is determined if there is an Rx33 controller responding at
                        ; address 177777400. If so, registers 177777402-406 are tested for stuck bits.
                        ; If this test fails, the register being accessed and found in error is placed
                        ; in the error table at memory address 400.
                        ;
                        ;       NOTE : Bits 0:7 of the Rx33 CSR (177777400) cannot be written since
                        ;       this is the command register, and we don't want to start the Rx doing some
                        ;       command.
                        ;
                        ;       INPUT  PARAMETERS :R0 has base address of partition just tested by memory.
                        ;           SP is set to top of this partition.
                        ;       OUTPUT PARAMETERS : If test fails control transfers to fault light routine.
                        ;           On success, execution falls thru to further Rx33 tests.
                        ;+-----------------------------------------------------------------------
                        ;       Register Assignments
                        ;
                        ; R0    : Base address of tested partition from memory program
                        ; R1    : data image to write/compare against
                        ; R2    : used for address of register under test
                        ; R3    : scratched. used for timeout
                        ; R4    : number of registers to test
                        ; R5    : data image count
                        ; R6    :
                        ;+-----------------------------------------------------------------------

 174732  005267  175402 RXRE6:  INC     TESTNO                  ; set test number to 4

 174736  012702  000004         MOV     #4,R2
 174742  012722  175342         MOV     #RXDN001,(R2)+          ; SET OWN NXM VECTOR, in ram
 174746  012712  000340         MOV     #340,(R2)               ; AND DISABLE INTERRUPTS, if NXM occurs

                        ; If you get here , there is an Rx33 out there, otherwise you would NXM trap.
                        ; Whether or not the Rx is healthy yet is open for debate at this point.

 174752  012702  177402         MOV     #RXCSR+2,R2             ; START OUT IN 'TRACK' REGISTER
 174756  012704  000002         MOV     #2,R4                   ; NUMBER OF REGISTERS TO DO

                        ;-- Begin --->
 174762  012705  000020 10:     MOV     #16.,R5                 ; NUMBER OF DATA PATTERNS TO TEST
 174766  012701  000001         MOV     #1,R1                   ; STARTING IMAGE

 174772  010112         20:     MOV     R1,@R2                  ; IMAGE--> REGISTER
 174774  004767  001600         CALL    WAIT                    ; wait gives you 33.2 us of slack time
 175000  021201                 CMP     @R2,R1                  ; COMPARE WAS WRITTEN WITH WHAT IS
 175002  001167                 BNE     BITSTK                  ; go to bit stuck report----> exit

 175004  006301                 ASL     R1
 175006  077507                 SOB     R5,20                   ; do this until we hit bit 15
                        ;-- End ----->

 175010  012205                 MOV     (R2)+,R5                ; ONE INSTRUCTION FOR INC R2 BY TWO
 175012  077415                 SOB     R4,10                   ; DO THIS FOR THREE REGISTERS
```

```
                        ; now all general registers are tested.  Verify some bits in the upper byte
                        ; of the CSR to make sure we have COMPLETE CONTROL.

175014  012701  011064         MOV     @#TRENA!DISRQ!verify,R1 ; image of what should be
175020  012702  177400         MOV     @#RXCSR,R2              ; address of RX controller
175024  032712  166000         BIT     #166000,(R2)            ; make sure no bits are stuck
175030  001154                 BNE     BITSTR                  ; EXIT IF STUCK BITS

                        ; when you fall thru to the next page general health of the Rx33 registers is
                        ; indicated.  Proceed with caution, and TERMINATE ON ERROR!!
                        ;+-------------------------------------------------------------------
```

```
                        .sbttl  ..TEST 4 PART II Interrupts & DMA
                        ;+------------------------------------------------------------------------
                        ;                       ROUTINE DESCRIPTION :
                        ; This test is to see whether the Rx33 can perform basic operations such
                        ; as a simple interrupt. DMA hardware is tested to see that address counters
                        ; increment properly, and that the data path to/from memory is error free.
                        ; All of the data path bits are checked , as well as the parity.
                        ;
                        ;               INPUT  PARAMETERS :
                        ;               OUTPUT PARAMETERS :
                        ;+------------------------------------------------------------------------
                        ;       Register Assignments
                        ;
                        ; R0    :
                        ; R1    : BIT mask to load DMA address
                        ; R2    : Address of register under test
                        ; R3    :
                        ; R4    :
                        ; R5    :
                        ;+------------------------------------------------------------------------
```

```
 1 175032  010201                RXCINT: MOV     R2,R1           ; R2 STILL HAS CSR ADDRESS (also data we want to use)
 2 175034  005722                        TST     (R2)+
 3 175036  010122                        MOV     R1,(R2)+        ; move 177400 to RXTREK
 4 175040  012767  176572  003162      MOV     #RXINT,RXIVEC   ; place the interrupt vector for forced interrupt
 5 175046  012767  000340  003156      MOV     #340,RXIVEC+2   ; move priority during interrupt
 6 175054  106427  000000              MTPS    #0              ; set priority to zero so we can be interrupted
 7 175060  010122                        MOV     R1,(R2)+        ; Move same thing to RXSEC
 8
 9 175062  004767  001512                CALL    WAIT            ; okay so it takes two extra words... it's safe
10
11 175066  012705  000003                MOV     #3,R5           ; basic error code in case of failure to interrupt
12 175072  012703  031330                MOV     #RDENA!DISRQ!FNCINT,R3  ; R3 has command to be able to force interrupt
13 175076  004767  001506                CALL    EXECOM          ; go and execute the interrupt command
14 175102  103543                        BCS     T4ER1           ; error = controller did not interrupt
15
16 175104  012767  011320  002266      MOV     #INTRENA!DISRQ!RSINT,RXCSR      ; reset after forced interrupt
17 175112  004767  001462                CALL    WAIT
18 175116  012767  176566  003104      MOV     #RXINS,RXIVEC           ; place regular interrupt handler
19 175124  012712  040000                MOV     #DMATST,(R2)    ; load test mode to RXDBR
20 175130  004767  001444                CALL    WAIT            ; WAIT FOR 33.2 us
21 175134  012703  021254                MOV     #RDENA!WRSEC,R3 ; this will be the WRITE command
22 175140  032767  000200  002232        BIT     #RDYRDY,RXCSR   ; verify that we do not have ready do to DMA mode
23 175146  001520                        BEQ     T4ER2           ; (DMA MODE DID NOT ISOLATE DRIVES)
24
25 175150  004767  001434                CALL    EXECOM
26 175154  103516                        BCS     T4ER1           ; no interrupt
27
28 175156  032712  000400                BIT     #BIT8,(R2)      ; see if we got the carry
29 175162  001511                        BEQ     T4ER3           ; counters wrong (DMA logic bad)
30
31 175164  030142                        BIT     R1,-(R2)        ; check for any upper order bits that shouldn't be
32 175166  001107                        BNE     T4ER3           ; counters wrong (DMA logic bad)
33
34 175170  032742  177000                BIT     #177000,-(R2)   ; check for any bits stuck in RXTRK
35 175174  001104                        BNE     T4ER3           ; bits stuck in count (DMA LOGIC BAD)
```

```
                                ; at this point the counters have been checked for the correct incrementation
                                ; now for fun and profit we will check the data path
                                ;
175176  005067  002204                CLR     RDMAR           ; remove DMA mode so we can load other registers
175202  004767  001372                CALL    WAIT
175206  005022                        CLR     (R2)+           ; zero the track register too
175210  004767  001364                CALL    WAIT
175214  012722  000125                MOV     #125,(R2)+      ; load sector w/data pattern; update pointer to RDMAR
175220  004767  001354                CALL    WAIT
175224  012712  040000                MOV     #DMATST,(R2)    ; set dma test mode in RDMAR
175230  012703  021214                MOV     #GOENA!RDSEC,R3 ; now do DMA read from disk to MEMORY
                                ;
                                ; now at this point if you get a PARITY ERROR when you check the result in
                                ; memory, you will trap thru 114 to the test error vector below.
                                ; Then you will end up with a fault light. If you get a parity error after the DMA
                                ; operation, there is no sense in recovery.
                                ;
175234  012737  175402  000114        MOV     #T4ER5,#0114    ; set up parity trap vector for checking
175242  004767  001342                CALL    EXECOM          ; do the command ( should write memory)
175246  103461                        BCS     T4ER1           ; (INTERRUPT DID NOT OCCUR)
                                ;
175250  005012                        CLR     (R2)            ; Remove DMA test mode; test result so registers settle
175252  022737  052525  000000        CMP     #052525,#0      ; see if data transferred (parity trap may take you here
175260  001051                        BNE     T4ER4           ; (DMA DATAPATH BAD)
                                ;
175262  012742  000031                MOV     #31,-(R2)       ; load pattern which will have different parity
175266  004767  001306                CALL    WAIT
175272  012767  040000  002106        MOV     #DMATST,RDMAR           ; set DMA TEST MODE
175300  012703  021214                MOV     #GOENA!RDSEC,R3         ; load read command
175304  005712                        tst     (r2)                    ; pad the loop
175306  004767  001276                CALL    EXECOM          ; should have settled by now
175312  103437                        BCS     T4ER1           ; no interrupt, (op not complete)
                                ;
175314  022737  014431  000002        CMP     #14431,#2       ; see if two has pattern
175322  001038                        BNE     T4ER4           ; if data not correct
                                ;
175324  012737  174638  000004        MOV     #MBADMEM,#4     ; move memory test vector back
175332  012737  174638  000114        MOV     #MBADMEM,#0114  ; if we get parity error after this.. bad memor.
175340  000430                        BR      ROOTMD          ; hardware is OK;  find the first working drive
                                ;+-------------------------------------------------------------------------+
                                ;
```

```
                        .sbttl  Register & hardware test error traps
                        ;+-------------------------------------------------------------------------+
                        ;
                        ;
175342                  NXMERR:
                        ;
                        ;       NXM hardware vector is set to here in case of a NXM when trying to get at
                        ;       the Rx33 registers.
                        ;       save register being addressed and NXM code in Rx error table; GOTO lights.
                        ;       we will leave the NXM vector set to here during the hardware test as well.
                        ;       we might catch intermittent failures that way, and it's free.
                        ;
175342  005302          ;       DEC     R2
175344  005302          ;       DEC     R2                      ; ADJUST R2, SINCE AUTO INC IS USED
175346  010267  003030  ;       MOV     R2,(ERTAB0+2)           ; save the address which gave NXM
175352  012767  000001  003020  ;       MOV     #1,ERTAB0       ; and deposit '1 when accessing Rx33' code
                        ;
175360  000416          ;       BR      RXCPN                   ; GOTO fault light; that's all we know
                        ;
175362                  BITSTR:
                        ;       if we find bad bits in the Rx33 registers we end up here
                        ;       We stuff the bad bit info as well as the address involved
                        ;
175362  012704  000400  ;       MOV     #ERTAB0,R4              ; ADDRESS OF TABLE
175366  012724  000002  ;       MOV     #2,(R4)+                ; set code = bad bit in register
175372  010224          ;       MOV     R2,(R4)+                ; SAVE OFFENDING REGISTER
175374  010124          ;       MOV     R1,(R4)+                ; EXPECTED BIT PATTERN
175376  011214          ;       MOV     (R2),(R4)               ; AND WHAT WAS REALLY THERE
175400  000406          ;       BR      RXCPN                   ; goto fault light routine
                        ;
                        ; This completes data entry into the error table. Xfer to common Rx33 error point.
                        ;
                        ;+-------------------------------------------------------------------------+
                        ; Below are the error traps for the DMA/INTERRUPT hardware tests.
                        ; Depending on your entry point, by the time you get to the move of R5 to the error
                        ; table in low memory, you will have incremented a number of times (0:4) which
                        ; will give you some unique error code word in the table. These are all hardware
                        ; faults of a fatal nature which would prevent booting by the Rx33.
                        ;+-------------------------------------------------------------------------+
                        ;
175402  005205          T4ER5:  INC     R5                      ; ERROR = 7     Parity error in datapath
175404  005205          T4ER4:  INC     R5                      ; ERROR = 6     Data wrong in memory
175406  005205          T4ER3:  INC     R5                      ; ERROR = 5     DMA address counters wrong
175410  005205          T4ER2:  INC     R5                      ; ERROR = 4     Test mode hardware fault
175412  010567  002762  T4ER1:  MOV     R5,ERTAB0               ; ERROR = 3     INTerrupt failed
                        ; just fall thru then to common gag point below.
                        ;
175416                  RXCPN:
                        ; the correct error information is already assumed to have been
                        ; stuffed in the error table.
                        ;
175416  000167  177206          JMP     RXCPN                   ; put a nonfat code in .. x   common fault
                        ;
                        ;+-------------------------------------------------------------------------+
                        ;
```

```
                        .sbttl  ..TEST 6 - Find a good Rx33 drive
                        ;+
                        ;**************************************************************************
                        ;
                        ;inputs:  R0 = Base address of the 0 Kword partition(supplied by memory test)
                        ;         MMU Kernel PAR 5 = Unit 0 of Rx33(lower byte) and type of boot (upper bit)
                        ;         MMU Kernel PAR 6 = CSR address of Rx33 to use for boot (16 bits)
                        ;         MMU Kernel PAR 7 = Validation word for unit and CSR
                        ;
                        ;
                        ;process:  Find a working Rx33 floppy drive can be used to read in the Init P.ioc test.
                        ;       A check is made to see if we were passed a specific Rx33 CSR and unit to use.
                        ;       Parameters in MMU Kernel PAR's 5, 6 and 7 are checked for validity by testing
                        ;       the validation word is equal to the XOR of the Unit & CSR words. If parameters
                        ;       are valid, the load is attempted via the Rx33 specified.  If parameters in the
                        ;       PARs are not valid, or if load attempt fails on unit specified, the default
                        ;       selection sequence (described below) is used.
                        ;
                        ;       DEFAULT SELECTION SEQUENCE (it's default of the drive)
                        ;       Either of the Rx33 drives that can be connected to the Mstd2 can be used as a
                        ;       boot device. Testing begins with Rx33 unit 0.  If unit 0 does not
                        ;       respond, unit 1 is tried.  If unit 1 also fails, we are in trouble.
                        ;
                        ;       Once we have a drive that we think works, we use 'RXREAD' within the ROM
                        ;       to load, or try to load, the first 8 blocks to the 0 Kword partition in
                        ;       Program memory.  If the first word of the loaded image is a NOP instruction,
                        ;       control is passed to the first instruction of the loaded image.  If the
                        ;       first instruction is not a NOP, then the next Rx33 unit in sequence is tried.
                        ;
                        ;       Failure to find any working Rx33 drive with media mounted  s a fatal
                        ;       error and results in the failure action listed below (see  outputs ).
                        ;
                        ;outputs:  Case 01 - A working Rx33 with bootable media mounted was found:
                        ;               Parameters passed:
                        ;               a) Type of boot (warm or cold)
                        ;               b) Base address of tested 0 Kword partition
                        ;               c) Unit number of the Rx33 that was used to load
                        ;               d) CSR address of the Rx33 that was used to load
                        ;
                        ;          Case 02 - No working Rx33 could be found:
                        ;
                        ;               State of J-11:
                        ;               a) 'Init' lamp is lit
                        ;               b) 'Fault' lamp is lit
                        ;               c) J-11 in loop monitoring the 'fault lamp'.  Depressing
                        ;                  the 'fault lamp' results in a blinking display of the
                        ;                  fault code for 'Load failure'
                        ;               d) J-11 must be rebooted to attempt recovery
                        ;
                        ;               Probable failure:
                        ;               a) No MSC70 system diskette is mounted
                        ;               b) Faulty Rx33 unit (if boot fails on just one unit)
                        ;               c) Faulty Rx33 controller (if boot fails on both units)
                        ;               d) Gamma rays
                        ;
                        ;               Recovery action:
                        ;               a) Mount an MSC70 system floppy (if none was mounted) and reboot
```

```
                        ;               b) Switch the MBC70 system floppy to the other Rx33 unit
                        ;                  (if proper media was mounted)
                        ;               c) Refer to description of Rx33 error table in the USEFUL INFORMATION
                        ;                  section of this listing
                        ;               d) Kick system, being careful not to injure you foot
                        ;
                        ;Notes:
                        ;
                        ;History of changes to this test:
                        ; Feb 1985 : changes to boot from Rx33. CSR to use left in for compatibility
                        ; Mar 1985 : more changes, and register test routine split out as separate test
                        ;*******************************************************************************
                        ;-
```

```
                                RXffud:
                                        inc     TESTNO          ; set test number to 6
                                        MOV     @#STDOR,R0#0R   ; set green LED on board; controller hardcore good
                                        ;+
                                        ;-------------------------------------------------------------------
                                        ;
                                        ;       Set the stack pointer (R6) to the highest location
                                        ;       tested.  Note that since we only load 8(10) blocks,
                                        ;       we avoid overwriting this area when reading from Rx.
                                        ;       Note - R4 always has the unit number to pass to the restore routine.
                                        ;-------------------------------------------------------------------
                                        ;-
                                        mov     R0,SP           ;point stack to lowest address of RX partition
                                        add     @#0A6SIZ,SP     ;adjust to last address (+2) of RX partition
                                        ;+
                                        ;-------------------------------------------------------------------
                                        ;
                                        ;       See if a RX unitNo and CSR were specified for booting
                                        ;
                                        ;       if (unit and CSR are valid) then ( try specified unit)
                                        ;       else ( use default selection sequence)
                                        ;
                                        ;       if (specified unit fails) then (use default sequence)
                                        ;       if (default sequence fails) then (PUNT)
                                        ;-------------------------------------------------------------------
                                        ;-
                                        mov     @#KDP#85,R4     ;R4 gets the unit word (upper bit=type of boot)
                                        bic     #100000,R4      ;clear out type of boot field
                                        mov     @#KDP#86,R2     ;R2 gets the CSR word
                                        xor     R4,R2           ;R2 gets XOR of unit and CSR words
                                        cmp     R2,@#KDP#87     ;is specified unit and CSR valid?
                                        bne     RXDEF           ;br if not and use default sequence
                                
                                        bic     #177400,R4      ;clear out upper byte of unit word
                                        mov     @#KDP#86,R1     ;R1 gets CSR address for Rx33 read subroutine use
                                        cmp     R1,#160000      ;insure CSR address in I/O page
                                        blo     RXDEF           ;br if CSR is not in I/O page, use default sequence
                                
                                        cmp     R4,#1           ;insure unit # is not greater than 1
                                        bgt     RXDEF           ;br if unit # is greater than 1, use default seq
                                
                                        CALL    RXDEST          ; See if requested drive can recalibrate and verify
                                        BCC     10              ; if test passed, try to boot from unit
                                
                                        CALL    ERRSAV          ; otherwise save the reason the default unit failed
                                        BR      RXDEF           ; then use default sequence
                                
                                10:     CALL    READBT          ;read boot blocks and check for bootable image
                                        bcc     trnsfr          ;br if read ok, and bootable image loaded
                                
                                        CALL    ERRSAV          ; save the failure to boot
                                
                                ; fall thru and try default selection sequence
```

```
                                ;******    DEFAULT  BOOT  SEQUENCE    *******

175540                          BOOET:

175540  005004                          CLR     R4              ;set unit 0 to zero

175542  012701  177400          10:     MOV     #RXCSR,R1       ; move address of CSR
175546  004767  000130                  CALL    RXREST          ; Go and attempt to do a recal/verify
175552  103007                          bcc     20              ; br if successful,to boot

175554  004767  000044                  CALL    ERRSAV          ; else, call error_save. R5=error code R4=unit number
175560  005204                          INC     R4              ; next try unitno 1
175562  020427  000001                  CMP     R4,#1           ; for unit=0, should be minus, for unit=1 should be 0
175566  101763                          BLOS    10              ; go back thru and attempt to recalibrate drive one

175570  000411                          BR      BOOFAL          ; BOOFAL is common exit point if no drives worked
                                                                ; Only this location can give a '23' failure
                                ;+
                                ;-------------------------------------------------------------
                                ;
                                ;       First try to read a bootable image from unit 0 of the
                                ;       currently selected controller.  If we can't load a bootable
                                ;       image from unit 0, try unit 1.
                                ;
                                ;       (If the read succeeds, we have read the boot blocks and the
                                ;        disk directory into the 8 kword partition.)
                                ;
                                ;-------------------------------------------------------------
                                ;-

175572  004767  000240          20:     CALL    READBT          ; read boot blocks and check for bootable image
175576  103015                          bcc     TRNSFR          ; br if read successful and image is bootable

175600  004767  000020                  CALL    ERRSAV          ; Otherwise save error code (13:17)

175604  005204                          INC     R4              ; point to next unit number
175606  020427  000001                  CMP     R4,#1           ; see if we have exceeded the number of drives
175612  101753                          BLOS    10              ; go back and execute test string for new drive

175614  012700  000023          BOOFAL: MOV     #LOADFL,R0      ; failure code for boot device failure
175620  000167  177810                  JMP     ANYFAL          ; goto common error routine

175624  110564  000410          ERRSAV: MOVB    R5,ERTABS(R4)   ; put in correct byte
175630  000207                          RETURN                  ; return to caller (saves code)
                                ;+-------------------------------------------------------------
```

```
                        .sbttl  ..TEST 7 - Transfer Control to Loaded Image
                ;+
                ;*****************************************************************************
                ;
                ;       The image just loaded to Program Memory is either an Init P.IOC test
                ;       or a non-HSC70 bootstrap (e.g., RT-11, RSX-11).  Either way, the image
                ;       must be massaged to work properly.  In the case of 'Init P.IOC' test,
                ;       a one block (512 byte) 'hole' exists between the first and second blocks
                ;       of the test, due to presence of a 'Volume ID' block on the Rx33.  This
                ;       hole is collapsed before transferring control to the loaded image.  In
                ;       case of a foreign (non-HSC70) bootstrap, the image must be moved from
                ;       the tested 8 Kword partition down to address 000000 to work properly.
                ;
                ;       After 'massaging' the loaded image, set up the parameters to be
                ;       transfer to the image.  For Init P.IOC test, transfer to the base address
                ;       of the loaded image. For a foreign bootstrap, transfer control to location 0.
                ;
                ;Inputs:        All tests passed (implied)
                ;               Rx33 parameters (CSR, Unit number)
                ;               Base address of working 8 Kword partition
                ;               Type of boot (warm or cold)
                ;               'Init' lamp is lit
                ;               'Fault' lamp is off
                ;
                ;Process:       Check the second instruction of the image just loaded
                ;
                ;               if ( second instruction = NOP ) then ( Init P.IOC test was loaded )
                ;               else ( a foreign bootstrap was loaded )
                ;
                ;               if ( Init P.IOC test was loaded)
                ;                       transfer to first address loaded)
                ;
                ;               if ( foreign boot was loaded )
                ;               then ( move image down to location 0, transfer to loc 0 )
                ;
                ;Outputs:       See 'Inputs' above
                ;
                ;Notes:
                ;
                ;History of changes to this routine:
                ; 4-Mar-85 compression no longer needed for Rx33, so this was removed
                ;*****************************************************************************
                ;-
```

```
                        TRNSFR: ;+
                                ;-------------------------------------------------------------
                                ;
                                ;       Set up the parameters that specify the RX33 unit and CSR used for
                                ;       the load just completed.  They may have changed if the unit
                                ;       specified for booting was not working.
                                ;
                                ;-------------------------------------------------------------
                                ;-
175632  005267  174582          inc     TESTNO          ;set test # to 7
175636  042737  077777  172352  bic     #77777,@#MAINT  ;mask everything except type of boot bit
175644  110437  172352          movb    R4,@#MAINT      ;save unit # we booted from
175650  010137  172354          mov     R1,@#MAINT+2    ;save CSR of Rx33 we booted from
                                ;+
                                ;-------------------------------------------------------------
                                ;
                                ;       Determine if we loaded the 'Init P.IOC' test or a foreign bootstrap
                                ;       if (second instruction of loaded image = NOP)
                                ;       then ( Init P.IOC test was loaded)
                                ;       else ( foreign boot was loaded )
                                ;
                                ;-------------------------------------------------------------
                                ;-
175654  010002                  mov     R0,R2           ;R2 gets base address of 0 Mword partition
175656  026227  000002  000240  10$:   cmp     2(R2),#000240   ;is second instruction a NOP?
175664  001005                  bne     50$             ;br if not, must be a foreign bootstrap
                                ;+
                                ;-------------------------------------------------------------
                                ;
                                ;       We have loaded the 'Init P.IOC' test
                                ;       The 7 blocks loaded are as follows:
                                ;
                                ;       block 0         first 512 bytes of 'Init P.ioc' test
                                ;       block 1         Not loaded
                                ;       block 2         second 512 bytes of 'Init P.ioc' test
                                ;       block 3         third 512 bytes of 'Init P.ioc' test
                                ;       block 4         fourth 512 bytes of 'Init P.ioc' test
                                ;       block 5         fifth 512 bytes of 'Init P.ioc' test
                                ;       block 6         first half of first RT-11 directory segment
                                ;       block 7         second half of first RT-11 directory segment
                                ;-------------------------------------------------------------
                                ;-
```

```
                                        ;+
                                        ;-------------------------------------------------------------
                                        ;
                                        ;       Set up parameters to be passed to the 'INIT P.IOC' test
                                        ;
                                        ;       R0 = Unit # of Rx33 used to boot, and 'type of boot'
                                        ;       R1 = CSR address of Rx33 (preserved for compatibility)
                                        ;       R2 = Base address of the tested 8 Kword partition
                                        ;       Priority is set to 7 for entry into just-loaded image
                                        ;-------------------------------------------------------------
                                        ;-
175666  016700  174460                  mov     @#UP0R5,R0      ;R0 gets unit # and type of boot
175672  016701  174456                  mov     @#UP0R6,R1      ;R1 gets CSR address of Rx33
175676  000112                          jmp     (R2)            ;transfer to the Init P.ioc test

                                        ;+
                                        ;-------------------------------------------------------------
                                        ;
                                        ;       The image just loaded is a foreign (non-MSC70) bootstrap.  It was
                                        ;       loaded to the 8 Kword partition in Program memory, but designed to
                                        ;       execute from locations 000000 thru 000776. So we move the first block
                                        ;       from the 8 Kword partition, down to address 000000, before jumping
                                        ;       to address 000000 to begin the bootstrap.
                                        ;
                                        ;-------------------------------------------------------------
                                        ;-
175700  010203                  50$:    mov     R2,R3           ;R3 gets base address of load
175702  005004                          clr     R4              ;R4 points to where to start moving data
175704  012705  000400                  mov     #400,R5         ;set R5 to count 1 block of data
175710  012324                  60$:    mov     (R3)+,(R4)+     ;move a word
175712  077502                          sob     R5,60$          ;branch until block is moved
                                        ;+
                                        ;-------------------------------------------------------------
                                        ;
                                        ;       Now jump to location 0 to continue foreign bootstrap
                                        ;
                                        ;-------------------------------------------------------------
                                        ;-
175714  116700  174432                  movb    @#UP0R5,R0      ;R0 gets unit number word
175720  005007                          clr     PC              ;jump to 0 to start foreign bootstrap
```

```
                        .sbttl  RXDEST- perform a RECAL/VERIFY operation

                        ;+
                        ;*****************************************************************************
                        ;
                        ;       Test intended unit number of Rx for ready/and track 0.
                        ; This is used as a test for the init code in the boot rom, as well as
                        ; a reposition routine every time a drive is accessed in the offline diagnostics.
                        ; It sort of replaces the old routine T50SYC, which synchronized with the TU58.
                        ; 15-Oct-1985 we use the fact that interrupts have been tested already to check
                        ; to do this RECAL command. Since this routine can be called from outside the Boot
                        ; ROM we 'steal' the interrupt vector and then replace it on exit.
                        ; The verify flag is set in addition to RECAL command, so the track register
                        ; gets loaded with the correct track at the end of the command.
                        ; If the drive is already at track 0 this routine takes no time at all.  Otherwise,
                        ; it can take up to 260Ms to pull in from the outermost cylinder. We allow 833 in
                        ; the command execute routine.
                        ;
                        ;inputs:
                        ;       R0 = Base address of tested partition in Program Memory
                        ;       R1 = CSR address of RX to use, passed for compatibility
                        ;       R4 = Unit number to test. (0 OR 1 in low byte)
                        ;       R2, R3, R5 are SCRATCH
                        ;
                        ;process:
                        ;       Steal INTERRUPT VECTOR for Rx33
                        ;       Give a RECALIBRATE to the drive
                        ;       Test for READY and NOT_BUSY from selected Rx drive
                        ;       Test for track 0. If not track 0, error out
                        ;       RETURN to calling program
                        ;
                        ;outputs:
                        ;       R0, R1, R4 are PRESERVED
                        ;       R2, R3 and R5 are CLOBBERED
                        ;       CARRY is set if ERROR or TIMEOUT occurred
                        ;       R5 will have a valid error code if CARRY is SET, else zero
                        ;*****************************************************************************
                        ;-
```

RXREST- perform a RECAL/VERIFY operation.

```
 1 175722                       RXREST::
 2
 3 175722                               PUSH    RXIVEC                  ; save current interrupt vector
 4 175726                               PUSH    RXPSW                   ; save priority
 5 175732  012767  176566  002270       MOV     #RXINT,RXIVEC           ; replace with ours
 6 175740  106427  000000               MTPS    #0                      ; set priority to zero now
 7 175744  012705  000010               MOV     #10,R5          ; basic error code, device not ready
 8 175750  012703  001320               MOV     #ENTRENA!RSINT,R3       ; zero/select the drive
 9 175754  004767  000630               CALL    EXECOM                  ; load command, expect timeout
10
11 175760  012703  031014               MOV     #RENA!DISRQ!RECAL!VERIFY,R3     ; the recal command to do
12 175764  004767  000620               CALL    EXECOM                          ; do the recal operation
13
14 175770  016703  001404               MOV     RXCSR,R3                ; get current image of CSR
15 175774  032703  000200               BIT     #RXRDY,R3               ; see if Not-ready is set
16 176000  001010                       BNE     TSer1                   ; door open, or no floppy (ERROR 10)
17
18 176002  032703  000030               BIT     #RNFERR!CRCERR,R3       ; check for RNF -or- CRC errors
19 176006  001004                       BNE     TSER2                   ; (ERROR 11)
20
21 176010  032703  000004               BIT     #TRACK0,R3              ; see if track_0 is set
22 176014  001003                       BNE     TSXiT                   ; if not, fall thru to (ERROR 12)
23
24 176016  005205              TSER3:   INC     R5                      ; ERROR = 12 TRACK_0 NOT SET
25 176020  005205              TSER2:   INC     R5                      ; ERROR = 11 CRC/RNF ON RACAL/VERIFY
26 176022                      TSER1:                                   ; ERROR = 10 DEVICE NOT READY
27 176022  000261                       SEC                             ; pass R5 to next level of software to process
28
29 176024                      tSxit:   POP     RXPSW                   ; restore PSW.  Preserve for external calling
30 176030                               POP     RXIVEC                  ; replace interrupt vector
31 176034  000207                       RETURN                          ; and return VIA here
32
33
```

```
                        .sbttl  Read Boot Blocks to Memory
                ;+
                ;*****************************************************************************
                ;
                ;       Read Rx33 boot blocks to Program Memory and test for Bootable image
                ;
                ;Inputs:
                ;       R0 = Base address of tested Program Memory partition
                ;       R1 = CSR address of Rx33 to be used
                ;       R2 = SCRATCH
                ;       R3 = SCRATCH
                ;       R4 = Unit # of Rx33 to be used
                ;       R5 = The LBN to read.
                ;
                ;Process:
                ;       Issue a RX33 read command for 1 block, load the data beginning at
                ;       the address in R0.
                ;       If that succeeded, issue a command to read blocks 2:8.
                ;       NOTE: Of the 8 blocks read, 5 are boot blocks, 1 is volume ID block,
                ;             2 are directory blocks.
                ;
                ;Outputs:
                ;       C clear if read is successful (no errors and first instruction = NOP)
                ;       C set if read fails or first instruction is not a NOP
                ;       R0,R1, and R4 are PRESERVED
                ;       R2 and R3 are CLOBBERED
                ;       R5 has failure code, if applicable
                ;
                ;*****************************************************************************
                ;-
176036          READBT: ;+
                        ;-------------------------------------------------------------------
                        ;
                        ;       call the Rx33 loader/read routine, (RXREAD)
                        ; This is the same routine that is called by the loader (later)
                        ;-------------------------------------------------------------------
                        ;-
176036                  PUSH    R0              ;save base address of partition on stack
176040  010002          mov     R0,R2           ;R2 gets starting address for load
176042  010400          mov     R4,R0           ;R0 gets unit number of Rx33
176044  012704  000001  MOV     #1,R4           ;R4 gets number to load first
176050  005005          clr     R5              ;R5 gets block number of 0
                ;call the read subroutine. R2 has updated VA
                ; R0,R1,R4 are untouched at return

176052  004767  000054  CALL    RXREAD
176056  103417          BCS     10$             ; branch if carry... failed first read

176060  012705  000002  MOV     #2,R5           ; otherwise get rest starting at block two
176064  012704  000006  MOV     #6,R4           ; R4 gets number of blocks to load following 0
176070  004767  000036  CALL    RXREAD          ; and get the remaining XXXX blocks of LBN 1
176074  103410          BCS     10$             ; if error on the transfer, then return
```

```
                                ;+
                                ;-------------------------------------------------------------
                                ;
                                ;       Check for a bootable image just loaded.  A bootable image
                                ;       contains a PDP-11 NOP instruction in the first word loaded.
                                ;
                                ;-------------------------------------------------------------
                                ;-

176076  010004                  MOV     R0,R4           ; restore unit number data to R4
176100                          POP     R0              ; Get old base address from stack
176102  021027  000240          cmp     (R0),#000240    ; is first instruction of image loaded a NOP?
176106  001407                  BEQ     20$             ; if so, then image 'should' be bootable

176110  012705  000017          MOV     #17,R5          ; error code for non-bootable image load
176114  000402                  BR      11$             ; punt out of here; set carry on the way out

176116  010004          10$:    MOV     R0,R4           ; if this path taken, return with carry set
176120                          POP     R0              ; and the old base address restored, just like

176122  000261          11$:    SEC                     ; the above code. R5 should have error code
176124  000207                  RETURN                  ; exit on error location

176126  000241          20$:    CLC
176130  000207                  RETURN                  ; RETURN with no errors.  Ready to start Boot

                                ;+-------------------------------------------------------------
```

```
                        .sbttl  RXREAD- read disk blocks to memory

                        ;+-----------------------------------------------------------------------
                        ;                       ROUTINE DESCRIPTION :
                        ;       RXREAD is the heart and soul of the bootstrap operation.
                        ; Here we attempt to load from the Rx, according to the parameters we are
                        ; passed in the general registers. This routine is also callable by the world
                        ; at large to permit further loading by INIPIO or OFLPIO.
                        ;
                        ; 20-Oct-85 : add clear of all but low bit of R0 ( unit # on entry)
                        ; -) This is for safety on a warm reboot
                        ;
                        ;               INPUT  PARAMETERS : Unit number,Virtual address to load to,
                        ;                                   # of LBNS to do, and lbn to start from
                        ;               OUTPUT PARAMETERS :  R0 restored. R5 has completion code
                        ;                                    R2 has updated Virtual address at end
                        ;                                   C bit clear if transfer successful
                        ;                                   C bit set if not. R5 has error code
                        ;+-----------------------------------------------------------------------
                        ;       Register Assignments
                        ;
                        ; R0    : UNIT # (passed by caller)
                        ; R1    :
                        ; R2    : Virtual address to transfer data to (passed by caller)
                        ; R3    :
                        ; R4    : Number of LBNS to transfer,(passed by caller)
                        ; R5    : LBN to start at (passed by caller) (error code on return)
                        ;+-----------------------------------------------------------------------

176132                  RXREAD::
176132                          PUSH    RXIVEC                  ; save old interrupt vector
176136                          PUSH    RXPSW                   ; save the PSW
176142  012767  176366  002060        MOV     #RXINS,RXIVEC           ; stuff address of boot ROMs interrupt handler
176150  042700  177776              BIC     #177776,R0              ; assure unit number okay on warm reboots

176154                          PUSH    R0                      ; SAVE ALL THESE
176156                          PUSH    R1
176160                          PUSH    R4                      ; save these


176162  106427  000000              MTPS    #0                      ; set priority to zero
176166  010403                      MOV     R4,R3                   ; play register game here
176170  010004                      MOV     R0,R4                   ; now R4 has unit, as it should
176172  010300                      MOV     R3,R0                   ; and R0 has number of LBNS to do

                        ;-- Begin ---) MAXIMUM OUTSIDE OF READ LOOP
176174  004767  000144        RXFRLP: CALL    RXJOAT                  ; sets up the Rx33 registers
176200                          PUSH    R5                      ; save R5 (lbn) use internal as error code
176202  012705  000013              MOV     #13,R5                  ; base error code internally
176206  103011                      BCC     RXLP1                   ; go ahead with transfer if seek not needed
```

```
                        ;+-----------------------------------------------------------------------+
                        ; Otherwise, pause on the read to data transfer and do a re-position (seek).
                        ; If the seek fails we abort before transferring any data.
                        ; Normally, on a BOOT operation SEEK will not be needed right away since the
                        ; first 8 blocks loaded are right off track 0, side 0. Later reads for rest of
                        ; BFLPIO, BMSPIO, and the LOADER will, in all probability, need to do seeks.
                        ; If the SEEK fails, decision to perform a retry rests on the calling routine.
                        ;+-----------------------------------------------------------------------+

176210  012703  021024  BOOSEEK: MOV     #BDEVA!SEEK!Verify,R3   ; Create command for seek
176214  004767  000370          CALL    EXECBM                  ; call the common command execute routine
176220  103437                  BCS     BOER1                   ; otherwise read seek error

176222  032767  000231  001150 PROBN:  BIT     #BDONERDY!BDFERR!CRCERR!BDBSY,BXCSR     ; any errors during seek?
176230  001032                  BNE     BOER2                   ; If so this is the 4th distinct error
                        ; If there were no errors, then proceed on thru with the read routine
```

```
                                ; Go forward with the transfer
                                ;
176232  012703  023214  RDLP1:  MOV     #RDCMA!LED!RDSEC,R3     ; set up the read sector command
176236  150103                 BISB    R1,R3                   ; or in the side info you saved from JOAT
176240  004767  000344         CALL    EXECOM                  ; and call the executioner
176244  103423                 BCS     RDER3                   ; error if command did not complete
                                ;
                                ; Test (PARITY_ERROR!NONEXIST_Ready)+(RNF)+(CRC)+(Lost_Data)
176246  032767  140235  001124 CHKST:  BIT     #RDCPE!RDNXM!RNFERR!CRCERR!RDHRDY!RDBSY!RDDLT,RDCSR
176254  001016                 BNE     RDER4                   ; This transfer was NOGO
                                ;
                                ; otherwise fall thru for (SUCCESS_OF_TRANSFER)
                                ;
176256                         POP     R5                      ; restore R5 (LBN)
176260  005205                 INC     R5                      ; point to next lbn
176262  062702  001000         ADD     #512.,R2                ; increment Virtual address
176266  005300                 DEC     R0                      ; Decrement (Loop_count)
176270  001341                 BNE     RDFRLP                  ; and return, in case >one LBN is requested
                                .
176272                 RDEX1:  POP     R4
176274                         POP     R1
176276                         POP     R0
176300                         POP     RDPSW                   ; restore old interrupt vector
176304                         POP     RDIVEC                  ; and old interrupt vector
176310  000207                 RETURN                          ; Normal, successful completion
                                ;
                                ; below is the error trap area for the read routine
                                ; just as in RDREST the error # you get depends on where in here you
                                ; enter
                                ;
176312  005205         RDER4:  INC     R5                      ; CODE = 16 hard read error
176314  005205         RDER3:  INC     R5                      ; CODE = 15 read timeout
176316  005205         RDER2:  INC     R5                      ; CODE = 14 seek error
176320                 RDER1:                                  ; CODE = 13 seek timeout
176320                         POP     R4                      ; first POP is really R5...which we discard
176322                         POP     R4                      ; instead let R5:=error code to go to next leve.
176324                         POP     R1
176326                         POP     R0                      ; cleanup stack
176330                         POP     RDPSW                   ; restore the PSW
176334                         POP     RDIVEC                  ; restore the old interrupt vector
176340  000261                 SEC                             ; set the error flag
176342  000207                 RETURN                          ; error exit; carry is set and table updated
                        ;+-------------------------------------------------------------------------
```

```
                        .sbttl  Rx33 Jack of All trades Subroutine
                        ;+-------------------------------------------------------------------------+
                        ;                       ROUTINE DESCRIPTION : ROJOAT calculates the physical
                        ; address on the disk of the upcoming transfer.  In addition, it figures out
                        ; what the DMA address of the upcoming transfer should be based on the virtual
                        ; address that is passed here. If the MMU isn't turned on, the virtual address is
                        ; the physical address as well, and no relocation calculations are needed.
                        ; The Rx33 controller registers are loaded at the same time with the DMA address
                        ; and the physical (sector/track) where we want to go on the disk. These are
                        ; loaded in this routine at the same time because it saves redundant register
                        ; accesses to the Rx33.
                        ;       INPUT  PARAMETERS : virtual address,LUN number
                        ;       OUTPUT PARAMETERS : carry set if seek required, R1 has side 1 value
                        ;+-------------------------------------------------------------------------+
                        ;       Register Assignments
                        ;
                        ; R0    : ROTRK image
                        ; R1    : ROSEC image. R1 has side code on return
                        ; R2    : RORQR image. Has Virtual address on entry
                        ; R3    : scratch, restored
                        ; R4    : scratch, restored
                        ; R5    : LUN on entry restored
                        ;+-------------------------------------------------------------------------+
```

```
                        ;
                        ;
176344                  ROLJOAT::
176344                          PUSH    R0              ; all of these are used
176346                          PUSH    R2              ; during the routine
176350                          PUSH    R3              ; all but R1 are restored on return
176352                          PUSH    R4              ; This is register-intensive but saves
176354                          PUSH    R5              ; time accessing Floppy registers
                        ;
176356  010203                  MOV     R2,R3
176360  005002                  CLR     R2              ; also assures that CARRY IS CLEAR
176362  032767  000001  001202  BIT     #1,@#MMR0       ; is MMU enabled?
176370  001426                  BEQ     BMINKT          ; if not, just go and load registers
                        ;
176372  010300          BMF16:  MOV     R3,R0           ; move it again
176374  042700  017777          BIC     #17777,R0       ; clear off all but upper 3 bits
176400  012701  000005          MOV     #5,R1           ; loop constant
                        ;
176404  006100          10:     ROL     R0
176406  077102                  SOB     R1,10           ; zoom ! rotate left 4 times
                        ;
176410  042700  177761          BIC     #177761,R0      ; only leave the 4 bit selection of PAR
176414  062700  172340          ADD     #KIPar0,R0      ; add 'base' PAR
176420  011000                  MOV     (R0),R0         ; get content of PAR (the relocate value)
176422  012701  000006          MOV     #6,R1           ; number of times to shift content (x 100)
                        ;-- Begin --->
176426  006302          20:     ASL     R2              ; remember, R2 was cleared earlier
176430  006300                  ASL     R0              ; here we simulate ASHC
176432  005502                  ADC     R2              ; prop carry if valid
176434  077104          30:     SOB     R1,20           ; loop around, and around, and around
                        ;-- End ----->
176436  042703  160000          BIC     #160000,R3      ; remove upper 3 bits of VA
176442  060003                  ADD     R0,R3           ; new physical address, simulate ADC
176444  005502                  ADC     R2              ; again propagate carry if it is valid
                        ;
                        ; fall thru to next page of code, where R2= highest 6 bits of 22 bit DMA address
                        ; and R3= low order 16 bits of virtual. Note that when the MMU is OFF we just
                        ; branch to the next page with R2 cleared and R3= 16 bit virtual address.
                        ;
```

```
 1 176446  016700  000730          DMINIT: MOV     DXTRK,R0
 2 176452  000300                          SWAB    R0
 3 176454  105000                          CLRB    R0
 4 176456  150300                          BISB    R3,R0
 5 176460  000300                          SWAB    R0                      ; R0 NOW HAS 'OLD TRACK' AND NEW DMA
 6 176462  010067  000714                  MOV     R0,DXTRK                ; wr.te as soon as possible, to let settle
 7 176466  ;;;;;                           MOV     R3,R1
 8 176470  105001                          CLRB    R1                      ; R1 is now waiting for 'sector' to load
 9
10 176472  000302                          SWAB    R2                      ; rotate R2 bits to upper DMA, if any
11 176474  052702  100000                  BIS     #BIT15,R2               ; We allowing full 22 bit DMA addresses
12
13                                  ; Now R0 has DXTRK, R1 has DCSEC and R2 has DDMAR
14                                  ; Next calculate the new track & sector values for the transfer and GO FOR IT
15
16 176500  005004          DSKPHY: CLR     R4
17 176502  071427  000036                  DIV     #percyl,R4              ; calculate 'TRACK'
18 176506  150402                          BISB    R4,R2                   ; R2 now has complete image to load
19 176510  010267  000672                  MOV     R2,DDMAR                ; so load it and let it settle
20 176514  005004                          CLR     R4
21 176516  071427  000017                  DIV     #persid,R4              ; next figure out sector
22 176522  005205                          INC     R5                      ; this is LBN bias. DO NOT REMOVE
23 176524  150501                          BISB    R5,R1                   ; R1 is finished, too
24 176526  010167  000652                  MOV     R1,DCSEC                ; load this , and let it settle
25 176532  005001                          CLR     R1                      ; use this for side parameter to return
26 176534  005704                          TST     R4                      ; see if upcoming transfer is on side 1
27 176536  001402                          BEQ     10                      ; if not, skip below instruction
28
29 176540  012701  000002                  MOV     #sidbit,R1              ; returned parameter needs ORed into command
30
31 176544  120002          10:     CMPB    R0,R2                   ; compare 'old track' and new
32 176546  001401                          BEQ     20                      ; skip below
33
34 176550  000261                          SEC                             ; set carry to indicate seek is required
35 176552                  20:     POP     R5
36 176554                          POP     R4
37 176556                          POP     R3
38 176560                          POP     R2                      ; notice R1 is not restored
39 176562                          POP     R0
40 176564  000207          NOJOAT: RETURN                          ; Now, all registers are set for transfer
41                                  ;+-------------------------------------------------------------------+
42
```

```
                        .sbttl  Rx33 Local Interrupt Handler
                        ;
                        ;+-----------------------------------------------------------------------
                        ;                       ROUTINE DESCRIPTION :
                        ;       We assume that if we get into this routine, that we got there via
                        ; an interrupt from the Rx33. A local location, INTFLG, is defined to always
                        ; be 177777 at entry to this routine. We use a zero for the flag.  In case the
                        ; dma engine gets the wrong address and writes over the INTFLG, it is unlikely
                        ; that it would write a zero there. This gives us an extra margin of safety.
                        ;
                        ;               INPUT  PARAMETERS : INTFLG = 177777
                        ;               OUTPUT PARAMETERS : INTFLG = 0;
                        ;+-----------------------------------------------------------------------
                        ;       Register Assignments
                        ;
                        ; R0    :
                        ; R1    :
                        ; R2    :
                        ; R3    :
                        ; R4    :
                        ; R5    :
                        ; R6    :
                        ;+-----------------------------------------------------------------------

176566  005767  000606        RX1MS:  TST     RXCSR                   ; read status register to knock down interrupt
176572  005037  000414        RXFIN:  CLR     @#INTFLG                ; clear to signal interrupts
176576  000002                RXRTI:  RTI                             ; and return from interrupt

                        ;+-----------------------------------------------------------------------
```

```
                        .sbttl  WAIT 33.2us Subroutine
                        ;+-------------------------------------------------------------------------
                        ; This subroutine has the sole purpose of wasting
                        ; 33.2us to make sure that the Rx33 registers have settled. It also wastes
                        ; R3 which is used for the count and is not restored.
                        ;+-------------------------------------------------------------------------

 8 176600  012703  000005      WAIT:   MOV     #5,R3
 9 176604  077301              10$:    SOB     R3,10$                  ; count of 5 in boot rom gives 33.2us delta-T
10 176606  000207                      RETURN                          ; exit
11                      ;+-------------------------------------------------------------------------
```

```
                        .sbttl  EXECOM - Common command interlock & execution routine
                        
                        
                        ;+-------------------------------------------------------------------+
                        ;                       ROUTINE DESCRIPTION :
                        ; EXECOM is a common routine shared by the hardware tests as well as the read code.
                        ; The command to be EXEcuted is passed in R3. R3 is then used for scratch (timeout).
                        ; The drive to execute the command on is passed in R4.
                        ;
                        ; 5-Nov-85 Add extra 'TST' instruction in wait loop for correct timeout in C-rev
                        ;               INPUT  PARAMETERS : R3 has command to do. R4 has unit (0 or 1)
                        ;               OUTPUT PARAMETERS : R3 clobbered , R4 preserved , carry set if fail
                        ;+-------------------------------------------------------------------+
                        ;       Register Assignments
                        ;
                        ; R0    :
                        ; R1    :
                        ; R2    :
                        ; R3    : command word to do. clobbered
                        ; R4    : unit 0. restored
                        ; R5    :
                        ;+-------------------------------------------------------------------+
                        
176610  000304          EXECOM: SWAB    R4              ; rotate the unit number to the high byte
176612  050403                  BIS     R4,R3           ; R3 now is complete with unit ID
176614  012737  177777  000414        MOV     #177777,@#INTFLG; set interrupt flag location to expecting interrupt
176622  010367  000552                MOV     R3,@#XXCSR      ; load the command ! here it goes !
176626  005003                  CLR     R3              ; start with 177777-1 for timeout count
                        ;-- Begin -->
176630  005737  000414  10:     TST     @#INTFLG        ; see if interrupt has occurred yet
176634  001406                  BEQ     EXEDNE          ; if so you were successful ?
                        
176636  005737  176776                TST     @#176776        ; This is to pad loop so timeout occurs
176642  077306                  SOB     R3,10           ; with the count, this entire loop times out in 833 ms
                        
176644  000304                  SWAB    R4              ; Fix R4 back up
176646  000261                  SEC                     ; set the error flag... SWAB clears carry
176650  000207                  RETURN                  ; this is 'bad' exit
                        
176652  000304          EXEDNE: SWAB    R4              ; restore R4 to previous state
176654  000207                  RETURN                  ; this is 'good' exit
                        
                        ;+-------------------------------------------------------------------+
```

```
                        .sbttl  Moving Inversions Test Description
                ;+
                ;*********************************************************************
                ;       Modified Moving Inversions ROM Test for PDP-11s
                ;
                ;TEST STRATEGY: Starting with the memory cleared to zeros, make 'n' passes (*)
                ;       from lowest address to highest address.  On each pass, every location is
                ;       read to verify that the last pattern was stored correctly, then the data
                ;       in the location is rewritten to contain an additional 'one'.  This is
                ;       continued until each location contains all ones.  Now that the memory
                ;       contains all 'ones', make 'n' passes thru the memory from highest address
                ;       to lowest.  This time read each location and verify that the previous
                ;       pattern is correct, replace a single '1' with a '0', and reread each
                ;       location to insure that the new pattern was stored.  When finished with 16
                ;       passes, the memory contains all zeros again.
                ;
                ;  (*) 'n' = number of binary bits per location
                ;
                ;Test Steps:
                ;       1)  Clear memory to all 'zeros'
                ;       2)  Starting at the LOWEST address, and working UP to the highest, read
                ;           a location and verify that the previous data was stored, replace
                ;           the location's data with a single additional 'one' bit, and reread
                ;           the location to insure that the new pattern was stored correctly.
                ;           This sequence is repeated until, on the final pass, the memory
                ;           contains all ones (16 passes thru the memory).
                ;       3)  Starting now at the HIGHEST memory address and working DOWN, read
                ;           each location and verify that the previous pattern is read cor-
                ;           rectly, rewrite the location's data to contain a single additional
                ;           'zero' bit, then reread the location to insure that the new pattern
                ;           was stored correctly.  Continue until the memory contains all zeros
                ;           (after the 16th pass).
                ;       4)  End of test
                ;
                ;FAILURE INDICATIONS: On each of the steps described above as a 'check', failure
                ;       to find the expected value is an error
                ;
                ;*********************************************************************
                ;-
```

```
                        .sbttl  Moving Inversions Test

                ;+
                ;*********************************************************************
                ;INPUTS:
                ;       R0,R1,R2,R3 are SCRATCH
                ;       R4 = Base address of the section of memory under test
                ;       R5 = Last address of the section of memory under test
                ;       R6 = Stack Pointer. (WARNING: do not push more than 2 words)
                ;
                ;OUTPUTS:
                ;       C Bit set if test FAILED
                ;               R0 = failing address
                ;               R4,R5 unchanged
                ;       C bit clear if test PASSED
                ;               R4,R5 unchanged
                ;               R0 thru R3 undefined
                ;
                ;REGISTER USAGE:
                ;       R0 = Address under test
                ;       R1 = Data Pattern to check for
                ;       R2 = Data Pattern to store
                ;       R4 = Base Address of memory under test
                ;       R5 = Last Address of memory under test
                ;       R6 = Stack pointer (WARNING: do not push more than 2 words)
                ;
                ;*********************************************************************
                ;-
176656          MOVINV: ;+
                        ;*********************************************************************
                        ;       STEP 1: Clear all memory locations to zeros
                        ;*********************************************************************
                        ;-

176656  010400          MOV     R4,R0           ;SET R0 TO BASE ADDRESS OF MEMORY TO TEST
176660  005020  100:    CLR     (R0)+           ;CLEAR A LOCATION(WORD), BUMP ADDRESS BY 2
176662  020005          CMP     R0,R5           ;CHECK FOR THE LAST ADDRESS CLEARED
176664  101775          BLOS    100             ;BR IF MORE TO CLEAR
                        ;+
                        ;*********************************************************************
                        ; STEP 2: Starting at the LOWEST address, and working UP to the highest,
                        ;       read each location and verify that the previous pattern is read
                        ;       correctly, rewrite the location's data to contain a single
                        ;       additional one, then reread the location to verify that the new
                        ;       pattern was stored correctly.  This sequence is repeated 16
                        ;       times, until the memory contains all 'ones'.
                        ;*********************************************************************
                        ;-

176666  005002          CLR     R2              ;INITIALIZE 'PATTERN TO STORE
176670  010201  200:    MOV     R2,R1           ;REPLACE THE LAST PATTERN WITH THE NEXT TO CHECK FO
176672  006302          ASL     R2              ;SET ONE MORE 'ONE' BIT IN R2 (PATTERN TO STORE)
176674  005202          INC     R2              ;(USING 'INC' VS. 'SEC' SAVES TIME)
176676  010400          MOV     R4,R0           ;SET R0 TO BASE ADDRESS OF MEMORY TO TEST
176700  020110  300:    CMP     R1,(R0)         ;CHECK FOR LAST PATTERN STORED SUCCESSFULLY
176702  001031          bne     1$0$            ;br if last pattern is not correct
176704  010210          MOV     R2,(R0)         ;STORE NEXT PATTERN
```

```
                        ;+
                        ;*********************************************************************
                        ;  STEP 3: Starting at the HIGHEST memory address and working DOWN to
                        ;       the lowest, read each location and verify that the previous
                        ;       pattern can be read correctly.  Rewrite the location's data to
                        ;       contain a single additional 'zero' bit, then reread the location
                        ;       to check that the new pattern was stored correctly.  Repeat this
                        ;       sequence until (on the 16th pass) the memory contains all zeros.
                        ;*********************************************************************
                        ;-

                        ;register state on initial entry to this step from previous step
                        ;       R2 = 177777,  R1 = 077777

176726 010201   600:     MOV     R2,R1           ;SET R1 TO 'PATTERN TO CHECK FOR'
176730 005700            tst     R0              ;(fastest way to clear the 'c' bit)
176732 006002            ror     R2              ;add another zero to pattern in R2
176734 010500            MOV     R5,R0           ;SET R0 TO HIGHEST MEMORY ADDRESS
176736 062700 000002     ADD     #2,R0           ;ADJUST ADDRESS 1st TIME SO WE CAN DECREMENT INITIALLY
176742 020140   700:     CMP     R1,-(R0)        ;CHECK FOR LAST PATTERN, DECREMENT ADDRESS BY 2
176744 001010            bne     1500            ;br if last pattern not stored correctly
176746 010210            MOV     R2,(R0)         ;STORE NEXT PATTERN (ONE ADDITIONAL 'ZERO-BIT')
176750 020210            CMP     R2,(R0)         ;CHECK FOR PROPER STORAGE OF NEW PATTERN
176752 001005            bne     1500            ;br if new pattern NOT stored correctly
176754 020004            CMP     R0,R4           ;CHECK FOR LOWEST MEMORY ADDRESS REACHED
176756 101371            BHI     700             ;BR IF STILL NOT AT LOWEST ADDRESS
176760 005702            tst     R2              ;when R2 = 0, the test is done
                        ;clc                    ;(side effect of clc instruction)
176762 001361            BNE     600             ;BR IF ANOTHER PASS THRU MEMORY IS REQUIRED
                        ;+
                        ;*********************************************************************
                        ;
                        ;       Exit
                        ;
                        ;*********************************************************************
                        ;-

176764 000207            rts     PC              ;exit with c bit clear

176766          1500:    ;(ERROR EXIT)
176766 000261            sec
176770          SCOPE:  ; these two labels are dummy vectors preserved for compatibilities' sake
176770          RECVPK: ; they both point to 'return instruction  below

176770 000207            rts     PC              ;return with c bit set
```

```
                        ;+
                        ;*********************************************************************
                        ;
                        ;       END OF P.IOC ROM BOOTSTRAP PROGRAM
                        ;
                        ;*********************************************************************
                        ;-

                                ;+
                                ;---------------------------------------------------------
                                ;
                                ;       'REMAIN' gets defined to be equal to the number of bytes left
                                ;       in the ROM (if positive), or to the number of bytes exceeded
                                ;       (if negative)
                                ;
                                ;       if (ROM size is exceeded by this program)
                                ;       then (issue assembly error)
                                ;
                                ;---------------------------------------------------------
                                ;-

000006                          remain=1aaddr-.+1       ;compute the # of bytes remaining or exceeded

                                .if lt remain-2
                                .error  ;ROM size exceeded by remain bytes
                                .iff
                                .=LAGAIN-1
176776  000403                  .word   PDPEDT+256.!PDPVER      ;low byte=ROM version, High byte=ROM edit
                                .endc

                        173000  .end     coldst
```

Symbol table

```
ALLUMP= 000037          DSUPNY  176500          MAXSEC= 000017          RDMSM   175342          SRV    = 021024
ALTBAT  173036          EMAXBP= 000200          MEMBDM  174616          RSINT  = 000320          SSRBAK  173574
ARRTAL  174634          ERRUPT  173026          MEMERR  174550          RXBSY  = 000001          STEP   = 000060
BADMEM  174630          ERRSAV  175624          MEMFAL= 000022          RXCINT  175032          STEPIN= 000120
BADSEC= 173000          ERTAB0= 000400          MEMRTY  174542          RXCMN   175416          STEPOU= 000160
BBSW   = 021000          ERTAB6= 000410          MEMTST  174414          RXCSR  = 177400          SUPCHR= 004000
BITSTR  175362          EXECOM  176610          MIMEMT  173024          RXDEF   175540          SUPFMT= 002000
BIT0   = 000001          EXEERR  176632          MODBAT  173732          RXDLT  = 000004          SWITST  174432
BIT1   = 000002          FLTLMP= 000010          MODDT1  173736          RXFAL   175614          TERMEM= 100000
BIT10  = 002000          FLTSW  = 004000          MOVINV  176536          RXFFWD  175422          TESTMD= 172340
BIT11  = 004000          FOO    = 001024          MSTDOK= 100000          RXFIN   176572          TRACKB= 000004
BIT12  = 010000          FRCINT= 000330          MTREMN= 001000          RXFRLP  176174          TRNSFR  175632
BIT13  = 020000          HALMP  = 000001          MULTI  = 000020          RXINS   176566          T4ER1   175412
BIT14  = 040000          HASH   = 000400          NXMBIT= 000004          RXIVEC= 000230          T4ER2   175410
BIT15  = 100000          HBLMP  = 000002          NXMJOB  176564          RXJOB   176344 G        T4ER3   175406
BIT2   = 000004          HBSW   = 001000          PARSIZ= 040000          RXLP1   176232          T4ER4   175404
BIT3   = 000010          HIPTST= 000040          PARTST  174456          RXMNR  = 177406          T4ER5   175402
BIT4   = 000020          INILMP= 000020          PERCYL= 000036          RXNRDY= 000200          TSER1   176022
BIT5   = 000040          INTFLG= 000414          PERSID= 000017          RXNXM  = 040000          TSER2   176020
BIT6   = 000100          INTIDX= 000324          PMEDT= 000001          RXOPE  = 100000          TSER3   176016
BIT7   = 000200          INTRDY= 000323          PMVER= 000003          RXOPSW  = 000232          TSXIT   176024
BIT8   = 000400          JMPTAB  173622          PIOFAL= 000021          RXREAD  176132 G        VECSIZ= 004000
BIT9   = 001000          JMPTR1  173636          POSDN   176222          RXRE6   174732          VERIFY= 000004
BLKLMP= 000004          JMPTR2  173638          PTIME0  174674          RXREST  175722 G        WAIT    176600
BLKSW  = 002000          JSRTST  174010          PTIME1  174714          RXRTI   176376          WRSEC  = 000254
BTYPE2  173010          J11ADC  174020          RDER1   176320          RXSEC  = 177404          WRTRK  = 000360
CHKST   176246          J11AD0  173644          RDER2   176316          RXTRK  = 177402          XAPR00= 172340
CLRBIT= 000100          J11DDN  174326          RDER3   176314          RX33RC  173030          XAPR01= 172342
CNTIE  = 000002          J11MTP  173742          RDER4   176312          RX33R0  173020          XAPR02= 172344
CRLDBT  173000 G        J11TST  173032          RDMDR  = 000304          RX33RN  173022          XAPR03= 172346
CPMFAL  174320          LASADD= 176777          RDSEC  = 000214          RX33SY  173016          XAPR04= 172350
CRCERR= 000010          LCLBIT= 000001          RDTRK  = 000340          SEEK   = 000020          XAPR05= 172352
DISRQ  = 010000          LED    = 002000          RDX1    176272          SEEKER= 000020          XAPR06= 172354
DIVTST  174266          LEDBIT= 000010          READBT  176036          SEEKL  = 000030          XAPR07= 172356
DMATST= 040000          LOADFL= 000023          RECAL  = 000010          SELP0  = 000400          XMMSR0= 177572
DMF16   176372          LODSIZ= 000010          RECVPK  176770          SELP1  = 001000          XMMSR1= 177574
DMINIT  176446          LOPTST= 000020          REMAIN= 000006          SENDPK  176770          XMMSR2= 177576
DOSEEK  176210          MAXCYL= 000117          RESETS= 000005          SIDE0  = 000000          XPIOCS= 170040
DRV0   = 000000          MAXLBN= 004502          RXFERR= 000020          SIDE1  = 000002          XSACSR= 170042
DRV1   = 000400
```

```
.. ABS.  177000    000   (RW,I,GBL,ABS,OVR)
         000000    001   (RW,I,LCL,REL,CON)
Errors detected:  0

*** Assembler statistics


Work file reads:  0
Work file writes:  0
Size of work file:  8359 Words  ( 33 Pages)
Size of core pool:  14336 Words  ( 56 Pages)
Operating system:  RT-11 (Under RTEM-11)

Elapsed time:  00:00:34.00
VS1:RDMSBC,VS1:RDMSBC=VS1:RDMSBC
```